Text detection in natural images using convolutional neural networks

by

Marco Marten Grond

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Applied Mathematics) in the Faculty of Science at Stellenbosch University

Supervisor: Dr WH Brink
Co-supervisor: Prof. BM Herbst

March 2017

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2017 .....

Abstract

In this study we attempt to solve the problem of text detection in natural images. This requires us to identify regions in a natural image that contain text. Possible applications range from assistive technology, human computer interaction and context extraction. Although humans find the task almost trivial, large variations in colour, font, size and orientation must be accounted for, and text shares many features and structures with other objects that cause complications when attempting to automate a solution.

We train multiple convolutional neural networks in an attempt to solve this problem. We chose convolutional neural networks both because they have already displayed potential in the context of text recognition, and to better understand how they operate. A sliding window approach is taken, where smaller regions of a full image are classified separately before the results are combined to identify text regions in the full image. Due to an insufficient number of annotated natural training images, we create a supplementary synthetic dataset. Using the synthetic data as a starting point we train networks of different structures, after which the same networks are finetuned on smaller natural datasets.

Networks first trained on the synthetic data outperform networks trained solely on the smaller natural datasets, regardless of structure complexity. This is likely due to an inability to identify relevant features from a limited number of training examples. Our experiments further show that a larger network structure is required for generalization, and that smaller datasets are prone to overfitting.

We apply our best performing trained network to the task of detecting text in full images, by extracting and classifying regions in an image using a sliding window. Image pyramids are also implemented to allow for greater variance in the size of text that can be detected. We find, however, that implementing image pyramids only slightly improves the accuracy over a single image, likely due to the fact that some scale variation was already present in the network's training set.

Ultimately, we find that convolutional neural networks show promise for the task of text detection in natural images. We also find that training a network on synthetic data and finetuning it on natural data improves the overall accuracy.

Uittreksel

In hierdie studie poog ons om teks in natuurlike beelde op te spoor. Die probleem vereis die identifisering van areas in 'n natuurlike beeld wat teks bevat. Moontlike toepassings sluit in ondersteuningstegnologie, mens-rekenaar interaksie en die onttrekking van konteks. Alhoewel 'n mens die taak baie maklik mag vind, moet variasies in kleur, lettertipe, grootte en oriëntasie in ag geneem word. Teks deel ook sekere kenmerke met ander beeldstrukture, wat die outomatisering van 'n oplossing verder kompliseer.

Ons poog om die probleem op te los deur verskeie konvolusie-netwerke vir die taak af te rig. Ons het besluit op hierdie soort neurale netwerke, aangesien hulle alreeds potensiaal in die konteks van teksherkenning getoon het, en ook om 'n beter begrip te ontwikkel oor hoe hulle werk. Ons onttrek kleiner vensters uit die beeld, klassifiseer elkeen afsonderlik, en kombineer dan die klassifikasies om areas van teks in die volle beeld te identifiseer. Vanweë 'n tekort aan geannoteerde data skep ons 'n aanvullende datastel van sintetiese beelde. Deur die sintetiese beelde as beginpunt te gebruik, rig ons verskeie netwerke met verskillende strukture af, waarna ons die netwerke met behulp van natuurlike data verfyn.

Netwerke wat eers op sintetiese data afgerig is vaar beter as dié wat slegs op natuurlike data afgerig is, ongeag netwerkstruktuur. Dit is moontlik te danke aan die feit dat 'n netwerk nie relevante kenmerke van teks uit min data kan identifiseer nie. Dit blyk verder uit ons eksperimente dat groter netwerkstrukture nodig is vir beter veralgemening, en dat kleiner datastelle oormatige passing tot gevolg kan hê.

Ons gebruik die beste afgerigte netwerk om teks in volle beelde op te spoor, deur vensters uit 'n beeld te onttrek en hulle te klassifiseer. Beeld-piramides word verder gebruik om die netwerke toe te laat om 'n groter variasie in die grootte van teks te kan identifiseer. Die gebruik van beeld-piramides het egter 'n klein impak op akkuraatheid, waarskynlik te danke aan die feit dat die netwerke reeds afgerig was op teks van verskeie groottes.

Deur die loop van hierdie studie het ons tot die gevolgtrekking gekom dat konvolusie-netwerke geskik kan wees om teks in natuurlike beelde op te spoor. Ons het ook gevind dat afrigting op sintetiese data en verfyning op natuurlike data die akkuraatheid van 'n netwerk kan verbeter.

Acknowledgements

I would like to express my sincere gratitude towards my supervisors Willie Brink and Ben Herbst for their continued support and guidance over the course of my Master's studies. Furthermore, I would like to thank the MIH MediaLab for providing funding and an enjoyable working environment, as well as the Harry Crossley Foundation for providing additional funding.

Contents

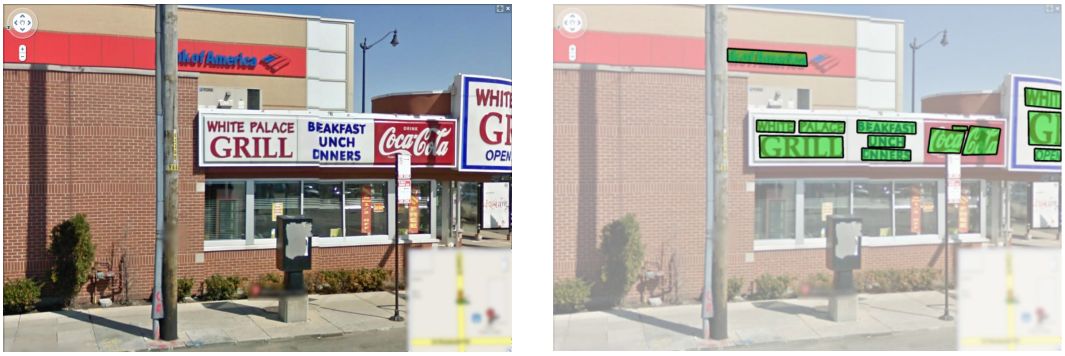
Chapter 1

Introduction

The aim of this study is to determine whether or not convolutional neural networks are viable for the task of text detection in natural images. These are images of natural scenes in which instances of text may occur, such as the example in Figure 1.1, as opposed to images taken for the sole purpose of text transcription (like scanned documents). Since convolutional neural networks often require large labelled datasets to properly generalize during training, we further investigate whether or not synthetic training data is a suitable replacement for large numbers of manually annotated natural images.

1.1 Text detection in natural images

Identifying text in natural images can be an important part of extracting information from images. Although humans find the task almost trivial, as is the case with many image classification problems, it can be complex for machines to identify features that define text. Cases where text appear on a busy background, or background structures that exhibit text-like characteristics can



(a) A natural image containing text. (b) Manually identified text regions.

Figure 1.1: An example of one of the natural images considered in this study. Source: Wang and Belongie (2010).

complicate the problem. Furthermore, text in natural scenes are subject to all the variabilities that may occur in natural images. Unpredictable lighting conditions, distortions, and variances in the orientation of text occur frequently and increase the difficulty of separating text from the background. In addition to all these problems, text may exhibit many different fonts, shapes and sizes, adding even more variability to account for.

The process of identifying and extracting text can be broken down into a number of steps. Initially, text has to be located in the image, then extracted from the image and, finally, the text has to be recognized (or transcribed) in order for meaning to emerge. We focus on the first part of this pipeline by attempting to identify regions in a natural image that contain text. Experiments performed by Bissacco *et al.* (2013) seem to indicate that increased accuracy in the first step of the process leads to increased accuracy of text recognition.

Our aim is to identify text in natural images, taking all of the previously mentioned variances into account. To accomplish this task we employ convolutional neural networks to detect areas in the image that might contain text. We opt for an approach of identifying regions containing text, rather than identifying single characters or strings of text, since convolutional neural networks are better suited to the task. Of course, strings of text can be extracted from identified regions using a variety of existing methods.

Our decision to use convolutional neural networks is twofold. Firstly, these networks seem to offer great promise for the task, and have been successful in past applications for text recognition (LeCun *et al.*, 1990), object detection (Krizhevsky *et al.*, 2012) and most importantly, text detection (Jaderberg *et al.*, 2014). The second reason is that these networks have become popular recently, and a subsidiary aim of this study is to explore and better understand how they operate.

Although much research has gone into text detection, there is still room for improvement. Many of the existing methods, which are discussed in detail in Chapter 2, rely heavily on predetermined thresholds and restrictions imposed on potential text candidates. We try to break away from the approach of thresholds and restrictions, and rather focus on learning abstract features that define text from raw data.

Neural networks are not limited by human feature engineering and during the training process learn to identify and extract features that define text and in some sense best separate text from other image content. For this reason it is important to have a large and representative training set, which is often difficult to acquire.

1.2 Applications of text detection

The main aim of accurate text detection is to extract information from an image, by enabling optical character recognition (OCR) software to efficiently

transcribe detected text. This is necessary for a wide variety of tasks, including assistive technology for people with visual impairments (Chen and Yuille, 2004; Yi and Tian, 2011), automatic translation between different languages (Haritaoglu, 2001; Chen and Dunsmoir, 2009), autonomous navigation and driver assistance (Ryzin, 1998), as well as extracting contextual information from images and videos (Li *et al.*, 2000).

By identifying and recognizing text in a natural image, additional information is extracted from the image. An example of how this could be useful, would be to allow people to translate text from a foreign language without the user first having to identify the text lines. Systems that can identify street signs or house numbers can also be used to assist users with navigation.

1.3 Objectives

The primary objective of this study is to investigate the viability of convolutional neural networks for the task of text detection in natural images. These networks require large datasets to generalize well, and our secondary objective is to explore the possibility of using synthetic data in the training of these networks, to compensate for a lack of publicly available, annotated natural data.

Although text recognition is one of the earliest applications of neural networks, we could not find existing work on text detection with neural networks at the start of this study. Lienhart and Wernicke (2002) used a related approach, where features are first extracted using other methods and then fed to a relatively small neural network for classification. More recently, Jaderberg *et al.* (2016) implemented convolutional neural networks to detect text in natural images, better approximate bounding boxes and recognize text extracted from natural images. Our aim is to investigate whether it is possible to locate text in raw image data, by considering smaller image regions and allowing our networks to learn relevant features from sufficiently large datasets (including synthetic datasets).

1.4 Our approach

To identify regions of text using neural networks, an input image can be broken down into multiple smaller windows which are passed through a trained neural network to be classified as either text or not-text. The network outputs a probability that each window is text, which can be mapped back onto the region that the window was extracted from to create a saliency map of the image, similar to the one in Figure 1.2, to indicate regions of text. This process can be repeated on various scales to produce a number of saliency

Figure 1.2: An idealized saliency map for the image in Figure 1.1. The map is blended over the original image, for visual orientation, and visualized in the standard jet colour scheme where red indicates high probability of text and blue indicates low probability of text.

maps, which can be combined and used to identify regions of text in the given image.

Neural networks often require large annotated training datasets to find acceptable solutions to a problem, that generalize well to previously unseen data. The creation of such training sets can be time consuming or subject to human error, however, if sufficiently large datasets are not readily available. For the training of our neural networks we consider synthetic data as a substitute for large annotated datasets of natural images containing text. In order to measure the effect of the synthetic data, we train multiple networks using different datasets and structures.

Although the approach of identifying and filtering connected components for text extraction seems to be more popular for this problem, we opt to use a sliding window since it does not have a heavy reliance on predetermined features or thresholds. Rather, we approach the problem with the intent of allowing the system to determine and use features that distinguish text from the background, without imposing our own ideas of what characterizes text. Although user input can be quite helpful in similar systems, it can also limit the potential of a system when features are overlooked or discarded. Neural networks are our chosen method of attempting this problem, for the very reason that they do not require predefined features to effectively classify data.

1.5 Contributions

In this study we show that convolutional neural networks can be successfully trained to locate text in raw image data, without the need for manual feature specification and extraction. We further show that by initializing a network on large amounts of synthetic training data, and then finetuning it on a much smaller set of real images, significant increases in detection accuracy can be

achieved. A portion of this work was published in the proceedings of a peer-reviewed international conference (Grond *et al.*, 2016).

1.6 Outline of the thesis

The next chapter of this thesis focusses on previous attempts to solve the problem of text detection in natural images. We examine multiple methods that we categorize into two different groups, namely the connected component based approach and the texture based approach. After giving some detail about their implementations, we draw conclusions of the different methods and compare them to one another.

In Chapter 3 we discuss the structure of convolutional neural networks and how they operate. We take a look at the different layers that comprise a network and the function of each. We also go into some of the finer details of training a network, after which we present a number of available software options for implementing these networks.

Chapter 4 focusses on our implementation. We initially go into some detail concerning the datasets that we use for training, by providing details of various existing natural datasets and explaining how we created synthetic images. Next we give details on our implementation of neural networks, explaining which structures are implemented and how the networks are trained. We end the chapter by giving an example of how a trained network can be applied to the task of text detection in a full image.

In Chapter 5 we present the results obtained from our implementation. We compare the different networks trained to one another and present our findings. The best performing network is then implemented for the task of identifying regions of text in full images. We also compare our results with other existing methods. We end the thesis by presenting conclusions and possibilities for future work in Chapter 6.

Chapter 2

Related work

Many different methods have been implemented in an effort to solve the problem of detecting text in natural images. In general, these methods can be classified as either connected component based approaches or texture based approaches. This chapter focusses on describing and comparing such previous methods. First, different methods employing either the connected component based approach or the texture based approach are described in detail. Afterwards, we examine the accuracy of each method and compare the methods to one another. Throughout, we also attempt to identify possible strengths and weaknesses of the different approaches.

2.1 Connected component based methods

Connected component based methods rely on finding homogeneous blobs of pixels that can be tested for text. These methods usually involve passing some algorithm over an image and grouping pixels together based on the output of the algorithm. Rather than searching for words or regions filled with text, these methods usually search for individual characters before grouping them together to form lines of text.

2.1.1 Contour based text detection

Yangxing *et al.* (2006) devised a connected component based approach for text detection by finding contours that define the outlines of text. This approach relies heavily on edge detection and using the detected edges along with their gradients to distinguish text from the background. Under the assumption that high contrast exists between the edges of text and the background, candidates for characters can be identified and evaluated to see if they conform to criteria describing text, before being combined together to form lines of text.

Initially an edge detector is passed over the image and each pixel is assigned a value of either 0 or 1, denoting non-edges and edges respectively. Text

edges are distinguished from other edges by an assumption of high contrast between edge pixels on text and neighbouring background pixels. Following this assumption, edges can be discarded if the magnitude of their gradient is lower than that of neighbouring pixels in the direction of the gradient. A threshold is also applied to remove weaker edges. The remaining edges are dilated to form continuous contours for the characters. In order for edges to be dilated and connected, the difference between their gradient directions must be less than some predefined threshold.

From the remaining edges, character candidates are computed. Following a left-right, top-down approach, each edge pixel is compared to the four previously encountered neighbouring pixels. Edge pixels are then grouped together into classes. If a pixel has a similar gradient magnitude to another neighbouring edge pixel, it is assigned the same class as that neighbour, otherwise it is assigned to a new class. For the case where multiple edge pixels fall within the allowed threshold, the pixel is assigned to all of the corresponding classes. The individual classes are all taken as character candidates.

Character candidates are filtered by applying some conditions. The first is that the average gradient magnitude of all the edge pixels should be greater than some predefined constant. Secondly, the variance between the gradient directions of the smallest and largest angles should also be greater than some constant. Finally, the number of edge pixels for a connected component must be larger than both twice the height and twice the width of the character candidate.

Each character is assigned a bounding box region, which is then put through a texture classifier to further weed out regions that do not contain text. The image is broken down with multiple low and high pass Haar wavelet filters, after which thresholds are applied to the second and third order wavelet moments, in order to eliminate not-text regions. Finally, the remaining character candidates are fed through OCR software in order to eliminate any remaining not-text regions.

This is the oldest connected component based method for text detection we looked at. The method uses a fairly straightforward approach and a set of clear characteristics that text is expected to exhibit. A possible downside to this method is that it might fail or perform poorly on text that differs from this preconceived notion of the appearance of text. Text that differs only slightly from the background might also be missed, while blurred images might obscure the distinction between text and background too much for the method to be able to differentiate between the two.

2.1.2 Stroke width transform

The stroke width transform (SWT), developed by Epshtein *et al.* (2010), is one of the best performing text detection techniques to date. It also employs edge detection to identify character candidates, but distinguishes itself from the

contour based method by grouping character candidates with similar stroke widths into chains.

The SWT technique assigns each pixel a stroke width, which is then used to group pixels together. The idea is that lines of text tend to exhibit uniform stroke widths across characters, and this property can be used to identify text in images. As such, the SWT is a connected component based approach. Since no OCR or statistics of gradient directions are used for detection, the approach allows for the detection of text in different languages and alphabets.

To compute the stroke widths of pixels in an image, a number of steps are needed. Initially an edge detection algorithm is applied to the image. The most commonly used algorithm seems to be the Canny edge detector (Canny, 1986) as used by both Epshtein *et al.* (2010) and Yao *et al.* (2012). Next, a gradient direction d_p is determined for every edge pixel p, with the assumption that if p lies on a stroke boundary, d_p should be approximately perpendicular to the orientation of the stroke.

Using the gradient direction, a ray $r = d_p n + p$ with $n > 0$ is followed until another edge pixel, q, is hit. If the two edges are more or less parallel, they should form the two boundaries of the stroke. The gradient of pixel q, d_q, is then compared to d_p. The two edges are seen to be approximately parallel if the gradient directions are roughly opposites of each other.

If two edge pixels are classified as two boundaries of a stroke, the stroke width values of all pixels along ray r are set to the distance between pixels p and q, unless a pixels already has a lower value. If no pixel q is found for pixel p, or if the gradients of the two pixels are not deemed to be opposites of one another, the ray is discarded.

Sometimes, especially along the corners of a character, the pixels in between two boundaries may have incorrect stroke width values. To account for this problem, another pass is performed for every ray not yet discarded, where all values along the ray that are above the median value along the ray, m, are set to m. This corrects outliers, as well as cases where the wrong ray was followed.

After the stroke widths of all pixels have been computed, they are then used to identify character candidates. Since the pixels of the image are now assumed to contain the widths of the most likely strokes they belong to, they can simply be grouped together into connected components. Neighbouring pixels are grouped together if their stroke widths are similar, or if the ratio of their stroke widths does not exceed some predetermined constant. Epshtein *et al.* (2010) found that a conservative ratio of 3.0 worked well for their experiments. This ratio also allows for natural variances in the stroke width of a text object as a result of font or perspective distortion.

The entire algorithm for detecting connected components is run twice, once for d_p and once for $-d_p$. This allows the algorithm to cover cases where the background is either lighter or darker than the text.

Once the connected components have been computed, the incorrect components are removed according to some rules (Epshtein *et al.*, 2010). Initially,

any component with too large a variance in the stroke widths of its pixels is removed (a variance of half the average stroke width for every component is used). This step removes inconsistent components such as foliage while keeping more uniform components such as text. Next, excessively long and thin components are removed, by limiting the aspect ratio of the components to be between 0.1 and 10. Furthermore, the ratio between the diameter and median stroke width of a connected component is limited to be less than 10. To eliminate components that surround text, such as frames, the bounding box of a component is limited to not include more than two other individual components. Finally, components that are excessively small or large are ignored. This effectively limits the allowed font size (a font range of 10 to 300 pixels is used). The remaining connected components are considered character candidates that can be grouped together into words and lines of text.

Grouping individual connected components into words further allows the algorithm to eliminate incorrect classifications, after making the assumption that isolated characters are unlikely to appear in an image. Since the font type and size are assumed to be uniform within a word, individual characters should have similar stroke widths, character widths and heights, while spacing between characters should be similar.

All pairs of character candidates are taken into consideration for belonging to the same line of text. Two characters are assigned to belong to the same text line if the ratio between their median stroke widths is less than 2.0 and they are sufficiently close to one another. Furthermore, a limit is imposed on the height variance between individual characters. The ratio of the character heights may not exceed 2.0, to allow some variance between the heights of upper and lower case characters. The width of space between two character candidates may also not exceed three times the width of the wider character candidate that is currently under consideration.

The next step involves clustering the individual character candidates into character chains. Initially, the chains consist of two characters that satisfy the above requirements. These chains can then be combined if they share character candidates and satisfy the above mentioned criteria. This process is repeated until no more chains can be combined. Finally, Epshtein *et al.* (2010) discard all chains that contain less than three character candidates.

Another implementation of the stroke width transform is one by Yao *et al.* (2012), and follows the above mentioned steps with a different set of rules for identifying text. The approach also considers text of a wide variety of orientations rather than mostly horizontal text.

The same steps as Epshtein *et al.* (2010) are followed up to the point where the connected components are extracted. Next, a two step filtering mechanism is implemented. Initially, the components are filtered by implementing a number of heuristics which discard components that fall outside some predetermined thresholds set out for them. For every component c, a width variation

WV(c), aspect ratio AR(c) and occupation ratio OR(c) are calculated as

$$\mathrm{WV}(c) = \frac{\sigma(c)}{\mu(c)} \tag{2.1a}$$

$$\mathrm{AR}(c) = \min\left(\frac{w(c)}{h(c)}, \frac{h(c)}{w(c)}\right) \tag{2.1b}$$

$$\mathrm{OR}(c) = \frac{n(c)}{w(c)h(c)} \tag{2.1c}$$

where $w(c)$ and $h(c)$ denote the width and height of the bounding box around c, $\mu(c)$ and $\sigma(c)$ the mean and standard deviation of stroke widths in c, and $n(c)$ the number of pixels in c. A component is discarded if $\mathrm{WV}(c) \notin [0, 1]$, $\mathrm{AR}(c) \notin [0.1, 1]$ or $\mathrm{OR}(c) \notin [0.1, 1]$.

The second part of the filter serves to remove not-text components that may have slipped through the first phase, regardless of scale or rotation. Using the Camshift algorithm (Bradski, 1998), the centre, scale and major orientation of each component are computed. A random forest classifier using these features assigns to each connected component a probability p_1 of being text.

Similar to the original implementation of the stroke width transform, Yao *et al.* (2012) group the character candidates together into chains. The process of computing character chains is adjusted by allowing non-horizontal chains to be created between components. The chains are combined using a greedy hierarchical agglomerative clustering method. A similarity measure is computed for chains that share at least one connected component, after which chains with the largest similarity rating are combined. The measure takes into account the angle between chains, as well as the number of candidates in each chain. This process is repeated until no chains can be combined, after which candidates not belonging to any chain are discarded.

Finally, the chains have to be filtered to remove any remaining not-text candidates. A random forest classifier is once again employed to compute a probability p_2 that the entire chain is text. This probability is used along with the previously computed probability of each connected component being text, p_1, to calculate the probability of the chain being a word. This is calculated as

$$p(C) = \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n} p_1(c_i) + p_2(C)\right). \tag{2.2}$$

Chains with probability $p(C)$ lower than some threshold are discarded. If a connected component belongs to more than one chain, it is simply assigned to the chain with the highest probability. This can happen, since the orientation of the chain is not bound to fall within certain limits, and characters from different lines of text could be combined.

The stroke width transform performs surprisingly well for something that seems quite obvious, yet was overlooked for a long time. It is quite a fast

method, since it does not require rescaling of the input image, and seems attractive for real time applications. One potential problem with the method is its heavy reliance on thresholds and tunable parameters, which could pose a problem for test cases that differ significantly from the training set.

2.1.3 Maximally stable extremal regions

Maximally stable extremal regions (MSER), first proposed by Matas *et al.* (2004), is an algorithm for defining blobs in an image, with a sliding threshold to combine pixels into larger blobs. The regions are determined by applying a sliding threshold, where pixels that have intensities higher than the threshold and are connected, are combined into extremal regions. An extremal region is defined as a region where the interior pixels have either a higher or lower intensity than the pixels on the border of the region. The maximal regions are those regions that remain more or less the same size for multiple thresholds. A tree can then be created where the smaller maximal regions are children of the larger maximal regions that encompass them. Such trees can be used to detect different objects in the image.

Chen *et al.* (2011) use the MSER algorithm to initially come up with multiple components that can be used as character candidates. Since the MSER algorithm was not originally conceived to detect characters, many of the character candidates may be incorrect.

The total number of character candidates is greatly increased by the fact that larger candidates can contain many smaller ones. Using the computed trees, many candidates can be removed by either pruning the child or parent nodes from the tree. The pruning takes place in two steps. Initially a linear reduction is applied where nodes in the tree with only one child are pruned, so that only the child or parent node remains. This reduction is applied recursively to the entire tree. Next, for each node with two or more children, the variation for different thresholds of each child is compared to that of the other child nodes as well as the parent node. The algorithm returns the child or parent with the lowest variation, so that only the disconnected, individual characters remain.

The remaining character candidates are combined into connected components using the single-link clustering algorithm, where character candidates with the shortest distance between them are connected into a new component. These components are once again considered for connection and the process is repeated until the distance between remaining components is larger than some threshold. The distance between two components is given as a function of the similarity between the components, which takes into account the stroke width variation and colour variations between components, as well as the physical distance between them.

Most of the connected components may not be text, however, and need to be removed. For each of the connected components the individual characters

are scrutinized by a classifier that takes into account stroke width features, the smoothness of the boundary and the height, width and aspect ratio of the character candidate. Using the classifier, each connected component is assigned a probability of being text, and all components with a probability lower than a certain threshold are removed.

The system is further extended by searching for connected components of arbitrary orientations. These components are then rotated to a more horizontal orientation and once again passed through the various steps of the system.

Yin *et al.* (2014) also use MSER features for text detection. They implement a Canny edge detector alongside MSER, and merge the results in order to more accurately distinguish characters from one another. The filtering of the MSER tree nodes places thresholds on the size and aspect ratios of the character candidates, and scrutinizes the stroke widths of the character candidates. The stroke widths are also used to combine the character candidates into connected components, since characters in the same word are assumed to have similar stroke widths. Finally, connected components are rejected if too much repetition or solidity is present within a single component.

Although the MSER method was not developed with text detection in mind, it seems to be capable of identifying characters. An obstacle would be cases where actual character nodes are pruned from the tree. This could lead to cases where partial characters are passed along, or even rejected by the system since they do not have characteristics of characters or text. Other text detection implementations that use some variation on the maximally stable extremal regions algorithm for text detection include the work of Neumann and Matas (2010, 2012).

2.1.4 Bounding box regression with convolutional neural networks

The most recent and best performing implementation comes from Jaderberg *et al.* (2016) who proposed a method of text detection and recognition using convolutional neural networks. Initially two weak algorithms — an edge box region proposal method as well as an aggregate channel feature detector — are implemented to find word bounding boxes. Combining these algorithms results in bounding boxes that cover most of the text areas, but may contain many false positives. The number of bounding boxes is reduced by implementing a random forest classifier, and finally the remaining boxes are passed through a convolutional neural network.

Bounding box generation is used in the first step, to circumvent the problem of different scales and to reduce the search space of the problem. The bounding boxes are generated regardless of scale and then filtered to discard false positives. To generate the bounding boxes that are to be tested, fast, high recall algorithms are employed. Even though the output may contain

many false positive boxes, this step reduces the search space for text. The edge boxes region proposal algorithm is combined with the aggregate channel feature detector to find the bounding box candidates.

The edge boxes region proposal algorithm (Zitnick and Dollár, 2014) uses the number of edges within a proposed bounding box to determine whether or not there exists an object within the box. Initially, an edge detection algorithm is applied to the entire image and the edges are grouped together into contours, defined as edges forming coherent boundaries. Multiple bounding boxes of different shapes and sizes are taken from the image and the number of contours contained within each box, without any intersection of its boundaries, are counted. All of the proposals are ranked based on the number, position and type of contours within the bounding box, as well as the shape of the box, and the best results are returned. This allows for thousands of proposals to be generated with high recall and low precision in a fraction of a second.

The aggregate channel feature detector is used to supplement the bounding boxes identified by the edge boxes algorithm. It uses feature pyramids, which are similar to image pyramids but require much less computational power and time. A number of different feature channels are computed for each image, which are then smoothed, divided into blocks, added up and smoothed again to give aggregate channel features. Multiple scales are then covered by computing feature pyramids. The time required to create the feature pyramids is reduced drastically by approximating the features for a specific scale. Features are computed at a single scale and multiplied by a scale specific constant raised to a channel specific power-law factor. A sliding window classifier that implements weak decision trees is finally applied to the different scales to produce bounding box candidates. This idea builds upon work by Dollár *et al.* (2014).

Taking the union of bounding box candidates from both the edge boxes and aggregate channel feature detector, an extremely high recall is achieved. Most of these bounding boxes are false positives, however, and need to be removed. Many of the correct bounding boxes also have a small overlap with the ground truth bounding boxes of the training data, and as such need to be corrected. The bounding boxes are filtered using a random forest classifier based on HOG features, that assigns a text or not-text label to each bounding box. A problem of the remaining bounding boxes, however, is that they often have insufficient overlap with the ground truth bounding boxes.

A convolutional neural network is used in order to correct the bounding boxes for a piece of text. A larger area, centred on the same pixel as the original bounding box but with twice the height and width, is taken as a replacement. The convolutional neural network is trained to regress the bounding box coordinates to ones that more closely resemble the ground truth bounding boxes.

Finally, the system employs another convolutional neural network to recognize and transcribe the text in the bounding boxes. Synthetic data is generated to train this network, and consists of thousands of different words with varying fonts and backgrounds.

While the weaker classifiers employed initially do not perform well on their own, their combination proves successful for detecting most of the text occurrences, which are then easily filtered by the much stronger neural network classifier. The use of fast detectors to identify possible lines of text, which each in turn reduces the number of examples, complements the stronger, slower classifier quite well. The use of a text recognition neural network should not be overlooked, as this could further improve the system by discarding any false positive detections.

2.2 Texture based methods

Texture based approaches assume that text has a different texture than the rest of the image content, and can be identified by partitioning the image based on texture. Using a sliding window approach, where small regions of the larger image are sequentially scrutinized, a saliency map is usually constructed to distinguish text from the background. Bounding boxes that identify lines of text can then be extracted from the saliency map.

The texture based methods are, in general, older than their connected component based counterparts. This is likely due to the fact that the texture based methods are slower because the input image has to be rescaled to account for varying text sizes.

2.2.1 Wavelet transforms

Li *et al.* (2000) use a combination of the Haar wavelet transform as well as a neural network to detect, and then later track text in digital videos. The texture based approach was chosen above the connected component approach due to the low resolution of the video frames, making it difficult to distinguish character candidates from one another and the background.

A 16×16 pixel sliding window is passed over the image with a step size of 4. For each window, the second and third order central moments are computed from the output of high-pass Haar wavelet filters applied to the window. Those moments are then used as input to a neural network which returns a probability of the window being text.

There is typically not much variance in the text that is sought, but more variance between the background possibilities. To account for this, the network is initially trained on a set of training examples. Afterwards, an image containing no text is fed through the system, and the areas classified as text are added to the training data with the correct label. This process is repeated multiple times, until the system achieves a desired accuracy.

When classifying all the pixels in a full image, multiple different scales are accounted for by implementing an image pyramid containing the same image in different sizes. A sliding window of constant size is passed over each image

and fed through the system. If the system classifies the window as text, all pixels within the window are classified as text. To combine the image pyramid classifications into a single image, areas classified as text are simply merged.

The system is further extended to track text across video frames. The text detection algorithm is only run at certain intervals to check for new occurrences of text, while the already detected text lines are tracked.

Another system that computes wavelet features in order to detect text is one by Ye *et al.* (2005), which uses the Daubechies wavelet transform. High-pass wavelet filters are once again applied to the image at different scales, and a wavelet energy feature is computed for each pixel. Pixels with an energy value higher than some threshold are taken as candidates for the rest of the algorithm.

A density based growing algorithm is implemented in order to extract text regions. Seed pixels, chosen based on the number of neighbouring pixels considered text candidates, are used to identify regions of text that are expanded to include text candidate pixels within a certain distance of the seed pixel. Any remaining candidate text pixels are discarded. The density based growing algorithm is applied at different orientations to account for rotated text. Constraints are also placed on the heights and aspect ratios of text lines to eliminate false candidates.

Due to a small number of training samples, a support vector machine is trained to classify the data based on wavelet features. Similar to the neural network implementation of Li *et al.* (2000), training of the SVM is repeated by adding new negative samples to the training set after training on a previous set has completed. Overlapping text regions detected at different scales and orientations are filtered by removing the candidate with the smaller probability or width.

2.2.2 Edge orientations with neural networks

A method for detecting horizontal text in images is presented by Lienhart and Wernicke (2002). An edge orientation image is computed for each input image, after which a sliding window is passed over it and fed through a neural network for classification.

After the edge orientation image has been computed, it is scaled to a variety of different sizes to accommodate different font sizes. Using a sliding window approach, 20×10 areas are taken from each scale and passed through a neural network to receive a number between -1 and 1, with all numbers above 0 considered text. The different scales are combined into a single saliency map by simply adding all pixel values above 0 to their corresponding pixels in the original image. This diminishes the effect that false positives have on the end result, as true positives are assumed to be detected at multiple scales.

A region growing algorithm is employed to generate bounding boxes, where pixels in the saliency map above a certain threshold are chosen as seeds. The

bounding box is expanded from the seed, by adding rows and columns that have an average value above another threshold. Using a projection profile for each bounding box, rows and columns within the bounding box are checked to see whether or not they contain text. This is then used to adjust the sizes of the bounding boxes to better fit the text, and eliminate incorrect bounding boxes.

2.2.3 Cascading AdaBoost algorithm

The AdaBoost algorithm has been used in many different object detection implementations. It works by combining different weak classifiers into a stronger classifier. Chen and Yuille (2004) use a cascading version of AdaBoost, where different features are checked at each level of the cascade, to detect text. A cascade was chosen to reduce the computational complexity of the system, since classifiers trained to detect simple features are used to discard obvious not-text samples before more complex features are computed.

The cascade consists of four layers with 1, 10, 30 and 50 classifiers respectively. The classifiers are simply trained on a single feature to return the probability of that feature being text, but do not have to have a high accuracy. The first three layers combine classifiers trained on simple features such as the intensity mean, standard deviation and derivatives. The final layer uses the more complex features, which include histograms of the intensities and edge linking. The AdaBoost algorithm uses a weighted combination of the probabilities computed by each individual classifier. The weights are determined through supervised training.

A sliding window approach is once again employed to analyse the entire image. Features are computed for each window as it is passed down the different levels of the AdaBoost algorithm, and windows that are classified as not-text are discarded as soon as they are classified as such. Text regions in the image are taken as the union of all windows that are classified as text by the entire algorithm.

2.2.4 Convolutional neural networks for character recognition

A texture based method that uses a convolutional neural network to identify characters is proposed by Wang *et al.* (2012). Although textures are not explicitly extracted from the image and then used to identify regions that may contain text, the convolutional layers of the neural network extract features from the input images, which are passed along and finally used to classify the input into one of the predetermined categories.

The convolutional neural network has the same structure as the classic one of LeCun *et al.* (1998), with two convolutional layers having 96 and 256

filters respectively, followed by two fully connected layers. Filters in the first convolutional layer are trained beforehand using K-means, and are not updated during the training of the network.

The system operates by passing a 32×32 window over the larger input image, once again using multiple scaled versions of the original image. Random 8×8 patches are extracted from these windows and used as input to the neural network, which should return a strong response when a centred character appears in the input. After the entire image has been passed through the neural network, non-maximal suppression is applied to horizontal rows in order to determine the positions of characters. Using this, bounding boxes are created for lines of text, dependent on the scale at which the characters were detected.

Jaderberg *et al.* (2014) implement a sliding window approach together with a convolutional neural network to determine bounding boxes for words, which are then identified using neural networks. Their convolutional neural network is trained to classify the bounding boxes as either text or not-text, but is limited by the scale of the text. As such, an input image is resized multiple times, and each time the sliding window classifier is passed over the image.

Using the sliding window approach, saliency maps are computed for each image and scale. Word bounding boxes are generated for an image at each scale, independent of the other scales. The probabilities in the saliency map are initially thresholded to find regions of high probability. Those regions are connected using the run length smoothing algorithm to produce lines of candidate text, which are then further split into words. This split is done to allow for easier character identification in the subsequent steps of the system.

2.3 Comparison

The two categories, connected component based and texture based text detection, both have their own strengths and weaknesses. The connected component based methods seem to be more popular in recent years, likely due to the speed with which an image can be processed. The accuracy of these methods is also quite high in most cases. The texture based methods on the other hand have been mostly ignored in the last few years, due to the computational complexity of having to use multiple different scales of the input image in order to account for different font sizes.

Text of different orientations and shapes continue to pose a problem for detection methods. Even though many of the above mentioned methods take different orientations into account when searching for text, it remains a challenging problem. Curved text is also problematic, especially for the connected component based methods, since most of them require character candidates to be more or less in a straight line. The texture based methods, on the other

hand, are not limited by this linearity assumption, as long as the textures they are trained on include text of varying orientations.

The connected component based methods do, however, have the upper hand when it comes to eliminating objects with similar structure to text. Since each component is considered as a character candidate, restrictions can be applied on the components to eliminate not-text objects. This allows the methods to eliminate objects that might exhibit text-like textures, but with incorrect structure. The texture based methods by comparison usually only use the aspect ratios as well as the height and width of the bounding boxes when attempting to eliminate such structures.

Extracting text from an image, for transcription purposes, might also be easier with connected component based methods. Many of the above mentioned connected component based methods output a black and white bit-map differentiating the character from the rest of the image, as opposed to the texture based methods that present bounding boxes. The extraction of bit-maps can be helpful for text recognition, since it eliminates background textures that could complicate the recognition process.

In terms of overall speed of execution, the connected component based methods again have the upper hand in most cases. This is mostly due to the fact that the algorithms do not need to be applied at different scales like the texture based methods. The fact that the texture based methods use a sliding window approach to search an image for textures means that without passing scaled up or down versions of the image through the system, text of differing sizes might be overlooked. However, with advances in the capabilities of hardware, parallel computing could allow text to be simultaneously detected at multiple scales.

Reported accuracies of the different connected component methods are presented in Table 2.1. The accuracies shown are all achieved on the ICDAR dataset (Lucas *et al.*, 2003), as reported by the authors. Most of the texture based methods were conceived before the ICDAR dataset, and as such the reported accuracies achieved on their own datasets are presented separately. It is also important to note that some of the methods are intended to be used for characters of multiple alphabets, while the ICDAR dataset contains only characters from the Latin alphabet.

To measure the accuracy of a text detection method, precision, recall and f-measure are used. The precision of an algorithm measures how accurate the detected bounding boxes are compared to the ground truth, while recall measures the percentage of ground truth bounding boxes that were detected. The f-measure is a combination of precision and recall. For two bounding boxes a measure m_p is calculated, which is defined as the area of intersection between the boxes divided by the area of the smallest bounding box that contains the original two bounding boxes, as illustrated in Figure 2.1. Furthermore, the best match for a bounding box r out of a set of bounding boxes R is defined

Method	Precision	Recall	f-measure
Epshtein *et al.* (2010)	0.73	0.60	0.66
Yao *et al.* (2012)	0.69	0.66	0.67
Yangxing *et al.* (2006)	0.64	0.67	0.65
Jaderberg *et al.* (2016)	0.96	0.85	0.90
Chen *et al.* (2011)	0.73	0.60	0.66
Yin *et al.* (2014)	0.67	0.85	0.75

Table 2.1: Comparison between precision, recall and f-measure of connected component based text detection methods, as reportedly measured on the ICDAR dataset (Lucas *et al.*, 2003).

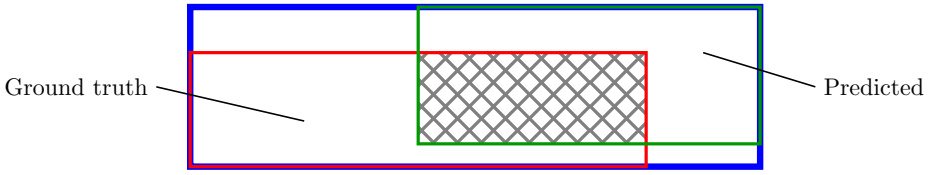


Figure 2.1: Overlap between the ground truth and predicted bounding boxes. The grey patterned area is the intersection between the two bounding boxes while the blue rectangle is the smallest bounding box that encapsulates both bounding boxes.

as

$$m(r_0; R) = \max\{m_p(r_0; r) | r \in R\}. \tag{2.3}$$

The precision, recall and f-measure are given as:

$$\text{Precision} = \frac{1}{|E|} \sum_{r_e \in E} m(r_e; T) \tag{2.4a}$$

$$\text{Recall} = \frac{1}{|T|} \sum_{r_t \in T} m(r_t; E) \tag{2.4b}$$

$$\text{f-measure} = \left(\frac{0.5}{\text{Precision}} + \frac{0.5}{\text{Recall}} \right)^{-1} \tag{2.4c}$$

where E is the set of predicted bounding boxes and T is the set of ground truth bounding boxes.

Regarding the texture based methods, Li *et al.* (2000) report that their algorithm correctly detected 261 out of 283 bounding boxes in their test set, with 23 false bounding boxes. Ye *et al.* (2005) measured a recall of 0.942 with a false positive rate of 0.024. Chen and Yuille (2004), after applying commercial OCR software to eliminate false positives, correctly detected 0.972 of text occurrences in their test dataset. Lienhart and Wernicke (2002) reported a recall of 0.695 on their test dataset. Finally, Wang *et al.* (2012) implemented an end-to-end text reading system, which after detecting text also attempts

to recognize the words. Their tests were conducted on the ICDAR dataset, and their best f-measure for word recognition is reported as 0.76. Jaderberg *et al.* (2014) tuned their network for high recall, and achieved a recall value of 0.80 with a precision of 0.19 on the ICDAR dataset. After filtering the results using word recognition, a precision, recall and f-measure of 0.89, 0.66 and 0.75 were achieved for full word recognition.

It appears as if the connected component based methods perform better than the older texture based methods. The texture based methods were likely abandoned in an attempt to increase the computational efficiency of the methods, resulting in research focussing on methods with scale invariance. A significant criticism against the connected component methods in general, is their heavy reliance on specific thresholds when classifying components as text. This could turn out to be a weakness that lets the system down when the text varies too much from the original training set.

The implementation of neural networks to solve the problem of text detection also seems quite promising, as demonstrated by Jaderberg *et al.* (2014, 2016). These networks, which are discussed further in the next chapter, learn features defining text, without user imposed thresholds or the requirement of a clear set of rules defining the expected appearance of text.

Chapter 3

Convolutional neural networks

Due to the recent success and popularity of convolutional neural networks, we decided to consider them for the task of text detection in natural images. They have already shown promise for related tasks like text recognition, as mentioned in the previous chapter.

Neural networks were first proposed in 1960, but fell into disuse due to the computational limitations of that time. With the appearance of more powerful computers, as well as the introduction of dedicated graphics processing units (GPUs) which allowed for larger networks and faster training times, neural networks have become quite popular recently. One of the most famous implementations of these networks is by Krizhevsky *et al.* (2012), which significantly outperformed previous attempts at image based object recognition. The reader is referred to Widrow and Lehr (1990) for a summary of the history of neural networks and how they work, as well as LeCun *et al.* (2015) for a more recent overview of neural networks.

A neural network typically consists of multiple layers that each performs a specific task. After the input array has been passed through a layer, the result is passed on to the next layer, where another operation is performed with the filters of that layer. This process is repeated in each layer until a classification vector is returned. This classification vector can be used to categorize the input array into one of the predetermined classes.

Convolutional neural networks are an extension of neural networks, and incorporate convolutional layers into their structure. These layers extract lower level features from the input, by dividing the input into smaller areas and effectively convolving the input with filters. The values of these filters are determined through training, where they are optimized to effectively recognize specific features corresponding to certain classes. If training is successful, the output of the network will be a vector with a strong response in the class that the input belongs to.

In this chapter we discuss the various aspects of convolutional neural networks. First, we explore the different layers that can be used to create these networks. Next, we take a look at possible algorithms for updating the net-

work weights and how the training process works. We then provide a list of readily available software options for implementing these networks. We end the chapter with some examples of where neural networks have been used.

3.1 Layers

Neural networks are made up of different layers that serve specific purposes. These layers are built up of individual neurons, that take input from the neurons in the previous layer and produce output by applying some function on the input. Different layer combinations as well as sizes (number of neurons in the layer) allow for a large number of different networks to be created. It is important to understand the function of each type of layer, since different combinations will lead to different results.

Deep neural networks, a term that has been widely used in recent years, simply refers to neural networks with multiple hidden layers. A hidden layer is any layer other than the input and output layers. The use of the term is a bit ambiguous, since some sources refer to any network with one or more hidden layers as a deep network, while others require a network to have at least three hidden layers to qualify. In the rest of this thesis we will simply refer to neural networks, regardless of the number of layers they contain.

Typically, convolutional neural networks consist of one or more convolutional layers, a number of fully connected layers and end with a suitable classifier like a softmax layer. The convolutional layers can incorporate different layers, such as the nonlinearity, sub-sampling and dropout layers. Although the neurons in the convolutional layer can implement a nonlinearity, sub-sampling or dropout, we have decided to present all these options as different layers (here we follow the convention of Caffe (Jia *et al.*, 2014), a software implementation discussed in Section 3.3.1).

3.1.1 Convolutional layer

The convolutional layers serve to reduce the dimension of the input array, by computing the dot products between $n \times n$ filters and the input array, which has been divided into a number of $n \times n$ regions known as receptive fields. The receptive fields are chosen by translating the filter, k steps at a time, until the entire input array has been traversed. The step size k is chosen so that $k < n$, for there to be some overlap between neighbouring receptive fields. An example of how a filter is applied to the input is shown in Figure 3.1.

The filters are used to extract simple features that are then passed on to subsequent layers, where they are combined and new filters can be used to detect more complex features. Due to the traversal of the filters over the input array, spatial displacement does not impede the network from locating and classifying an object.

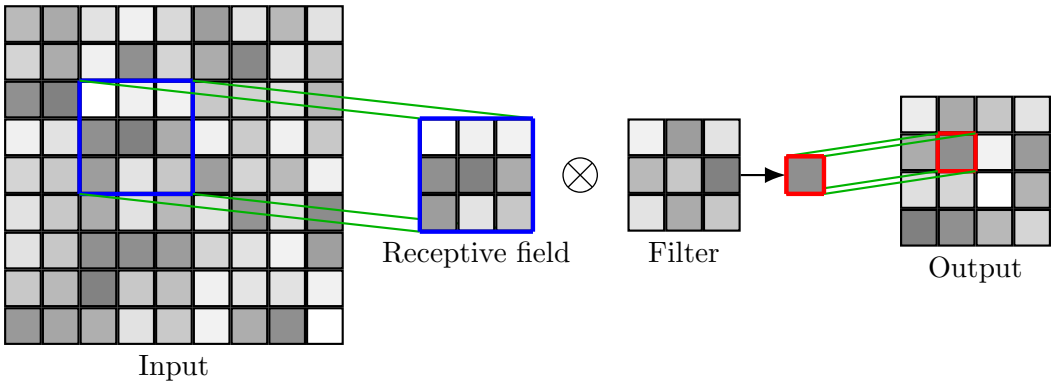


Figure 3.1: A 3×3 filter is applied to an input image of size 9×9 with a step size of 2. Here the symbol $\otimes$ denotes the operation where every value in the receptive field is multiplied by the corresponding value in the filter, after which the average of the results is given as the output of the function.

In practice, the filters are implemented by applying a plane of a single filter to the input array, instead of applying the filter to every receptive field consecutively, and receiving a feature map. The filters in these feature map planes have shared weights and biases, which cause the result to be the same as if a single filter was moved over the entire array. This weight sharing approach also reduces the number of parameters that need to be trained in the network. Multiple feature map planes exist in each convolutional layer in the network. Each plane is applied to the input array and the resulting feature map is passed on to the subsequent layers.

The result of applying a feature map plane to an array is a smaller array where the relative locations of features to one another are kept. The exact location of the feature is lost, due to the overlapping receptive fields and the step size between the receptive fields. In general the exact locations of features are not important, however, since the goal of the network is usually to classify the input into one of the predetermined classes (LeCun *et al.*, 1990, 1998, 2010; Krizhevsky *et al.*, 2012).

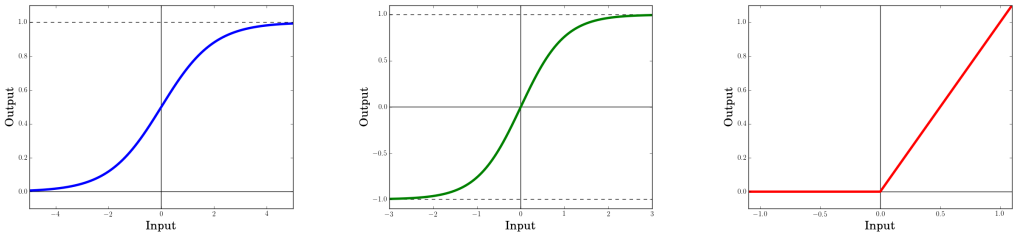
3.1.2 Nonlinearity layer

Single neurons, the building blocks of a neural network, all compute a linear function to map the input to the output. Since a linear function of linear functions remains linear, the network would be unable to model anything other than a linear relationship between the input and output. Adding a nonlinearity layer to the network enables it to map a function that may fit onto the problem space better than a simple linear function would be able to. Leshno *et al.* (1993) show that choosing a nonlinear, non-polynomial function allows the network to model virtually any possible function onto the problem space. The

chosen function has to be differentiable, since the derivative of the function is used when training the network.

In the past, smooth nonlinear functions such as $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$, depicted in Figure 3.2, were used to introduce nonlinearity in the network. Recently, the rectified linear unit (ReLU), $f(x) = \max(0, x)$, has been most popular, due to the fact that it allows for faster training of large networks. The sigmoidal functions tend to have gradients near 0 as the functions near their asymptotes. On the other hand, the ReLU has a gradient of 1 when the unit is activated and 0 otherwise, allowing for training to always take place if a unit is activated (Krizhevsky *et al.*, 2012; Maas *et al.*, 2013; LeCun *et al.*, 2015).

Since the derivative of the ReLU is 1 for all positive values, training will take place in a neuron as long as some input produces a positive output from the neuron. This removes the need to normalize the input of the nonlinearity layer, as is necessary for some of the sigmoidal functions. Krizhevsky *et al.* (2012) found, however, that applying a local response normalization after the ReLU aids generalization and reduces error rates. This is implemented by creating competition between the outputs of a neuron returned by the different filters for each receptive field.



(a) $f(x) = (1 + e^{-x})^{-1}$ (b) $f(x) = \tanh(x)$ (c) $f(x) = \max(0, x)$

Figure 3.2: Examples of nonlinear functions used in the nonlinearity layers of a neural network.

3.1.3 Sub-sampling layer

Sub-sampling is used to reduce the size of the output feature map that is passed on to the following layers, as well as to reduce the sensitivity of the output to displacements and distortions. This is achieved by reducing the resolution of the feature map in one of a number of possible ways.

The feature map is reduced into regions of size $s \times s$ and are spaced z indices apart. When $s = z$, the regions do not overlap, but when $z < s$ some overlapping occurs. Historically, most sub-sampling was performed without overlapping regions, but Krizhevsky *et al.* (2012) assert that some degree of overlap could reduce over-fitting.

These regions are then reduced and mapped to a new output feature map which is passed to the subsequent layers in the network. This reduction can be accomplished by taking the average over all values in the region, taking the maximum value or by using a stochastic pooling method which also seems to reduce over-fitting (Zeiler and Fergus, 2013).

Sub-sampling layers are usually paired together with convolutional layers, reducing the feature map produced by the convolutional layer before it is passed on further in the network. As demonstrated by LeCun *et al.* (1998), this reduction in size of the feature maps is necessary to reduce the size of the data passed through the network, since the number of feature maps are increased after each convolutional layer.

3.1.4 Fully connected layer

In a fully connected layer a neuron receives input from all neurons in the previous layer, as opposed to the convolutional layer where a neuron receives only the output of its receptive field in the previous layer. Each neuron in a fully connected layer therefore has a much larger set of weights than their counterparts in the convolutional layers. A simplified example of a fully connected layer is presented in Figure 3.3.

The fully connected layers usually form the final layers of a neural network and are used to identify the higher level features of the initial input to the network. This is accomplished by optimizing a function over the entire input, rather than a small region of the input.

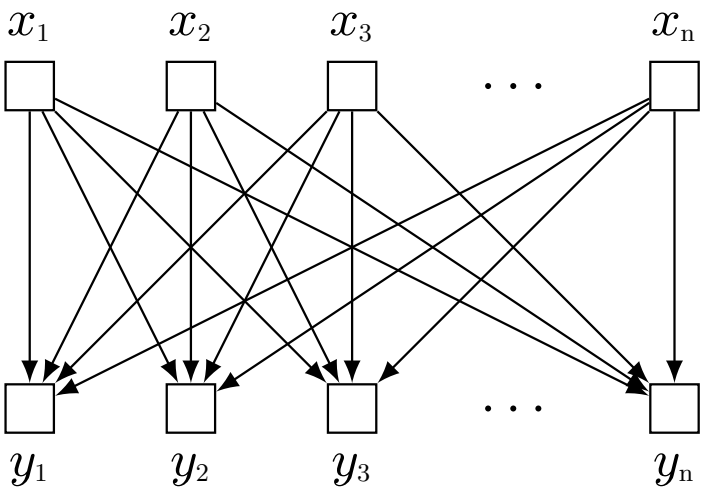


Figure 3.3: A simple demonstration of how a fully connected layer receives input. Each neuron in the fully connected layer receives the output of every neuron in the previous layer as input.

3.1.5 Dropout layer

In general, when testing a neural network on a holdout set, the accuracy obtained is less than would be obtained on the training set. This tendency towards over-fitting is reduced by implementing dropout. Dropout is applied by setting the output of each neuron to 0, with probability p, thereby possibly excluding it from classification and training during a particular forward pass. By implementing dropout, the dependencies between neurons are reduced since a neuron cannot rely on the presence of another specific neuron.

Implementing dropout is an efficient way of simulating the training of multiple networks and averaging their outputs. Since each forward pass effectively leaves out certain neurons, the architecture of the overall network differs for each training iteration. The neurons still share the same weight vectors. At test time, the entire network is used for classification, but the output is multiplied by p (Krizhevsky *et al.*, 2012; Hinton *et al.*, 2012).

3.1.6 Softmax layer

The softmax layer implements a generalization of logistic regression in that it is able to handle multiple classes. Where logistic regression is able to categorize its input into one of two classes, a softmax regression classifies the input into any number of classes, while at the same time normalizing the output to add up to 1. This allows for a more descriptive output, since each class is assigned a probability that the output belongs to it, rather than the network just returning the class with the strongest response.

The softmax layer usually receives multiple inputs that need to be reduced to the number of classes. This is done similarly to the fully connected layers, where a weight matrix is applied to the input which results in a vector with length equal to the number of classes. The values in this vector can be quite large or small, making it difficult to interpret the numbers. These numbers are normalized and presented as the output of the network. The loss, which is used for training the network, is computed using the expected output of the network and the output of the softmax layer. The probability of a class i in a network with n classes is given by

$$p_i = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \ldots + e^{z_n}}, \tag{3.1}$$

where z_i returns the i-th entry in the vector that is the dot product between the inputs into the softmax layer and the weight matrix of the softmax layer.

Since the softmax function is a generalization of the logistic function, for the case where n is equal to 2 the softmax function reduces to the logistic regression function.

3.2 Training

One of the main advantages of neural networks is the fact that their parameters are trained on annotated raw data, rather than human engineered features. During training, the network performs both a forward and backward pass. The forward pass consists of passing input data through the different layers of the network, and receiving the output vector. The network is trained during the backward pass, where the difference between the output vector and the expected output for the particular input is used to update the weight variables in the various layers. The metric used to measure the difference between the actual and expected output is called the loss and is computed in the last layer of the network. The loss is passed to previous layers in order to update their weights in a process known as back-propagation. Thousands of iterations are usually required to reach a good optimum, which can take several days to complete for large networks. There are also many training variables and methods for updating the weights in each layer of the network, that determine the speed of training as well as whether or not the training is successful (converges to a good optimum).

Before training can commence, the network parameters need to be initialized. Usually, random distributions are sampled to initialize the weights. There is an alternative, however, in the form of a process called finetuning. Weights of a previously trained network of the same structure are used to initialize a new network, before training through back-propagation proceeds as above. The adjustments to the weights of a network initialized in this manner are usually smaller than when training a randomly initialized network, since we want to retain and optimize the features detected in the previous network.

Most software options allow the user to implement batch training. This refers to a process where the network weights are only updated after a specific number of inputs, known as a batch, have passed through the network. The average loss for a batch is used when updating the weights. Wilson and Martinez (2003) show that both batch and the opposing method, on-line training, seem to present similar results for sufficiently large datasets. Batch training removes some noise from the gradient, while on-line training is able to follow the gradient curve more closely along the problem space.

Neurons in the network also have a bias weight, that always takes an input of 1. The weights of the bias are updated during training and allow the network to shift the classification function on the problem space by adding a constant value. This implies that the output of a neuron is given as

$$x_j = \sum_{i=1}^{m} (y_i w_{ji}) + w_{j(m+1)}, \tag{3.2}$$

where y_i refers to the various different outputs of the previous neurons with their corresponding weight vectors w_{ji}, $w_{j(m+1)}$ is the weight vector for the bias term and m is the number of neurons in the previous layer.

The overall aim of training is to minimize the loss function, since this will lead to an output vector that is closer to the desired output vector for the particular input. The loss, however, is only computed for the final output layer of the network, but all the weights in the previous layers need to be optimized. Rumelhart *et al.* (1986) give the loss as

$$E = \frac{1}{2}\sum_j (y_j - d_j)^2, \tag{3.3}$$

where j runs over all the individual neurons in the output layer, y_j is the actual output of neuron j and d_j is the expected output for the specific input case. In order to minimize the loss using gradient descent, the partial derivatives of E with respect to all weights used during classification need to be computed.

Initially, the partial derivative of the loss with respect to each output neuron is taken:

$$\frac{\partial E}{\partial y_j} = y_j - d_j. \tag{3.4}$$

The chain rule is then applied to compute the partial derivative of the loss with respect to the weights of the neurons in the previous layer. This is given as the weighted sum of the loss derivative with respect to the input of all connected neurons in the following layer:

$$\frac{\partial E}{\partial y_k} = \sum_j w_{kj}\frac{\partial E}{\partial z_j}, \tag{3.5}$$

where j refers to the connections between neuron y_k and the neurons in the following layer, w_{kj} refers to the weight vector applied to input neuron y_j from neuron y_k and $\frac{\partial E}{\partial z_j}$ is the partial derivative of the loss with respect to the input of neuron y_j (LeCun *et al.*, 2015).

Equation 3.5 can be used to compute the derivative of the loss function with respect to each neuron in the network. Once these derivatives have been computed, they can be used to update the weights of the individual neurons in the network. There are a number of different techniques for updating the weights.

Bottou (2012) advocates using stochastic gradient descent when updating the network weights, since it simplifies gradient descent for improved training times. The derivative of the loss of the network, scaled by some learning rate, is simply subtracted from the previous weight of the network. Furthermore, momentum, which takes the previous weight updates into account, as well as weight decay, which regularizes weights and prevents the weights from becoming too large, are added. This leaves us with the following scheme:

$$v_{t+1} = \mu v_t - \alpha \nabla E(w_t), \tag{3.6a}$$

$$w_{t+1} = w_t + v_{t+1} - \lambda \alpha w_t, \tag{3.6b}$$

where α is the learning rate, μ the momentum, λ the weight decay, v_t the previous weight update, w_t the current weight of the neuron and $\nabla E(w_t)$ the gradient of the loss function.

Stochastic gradient descent allows for the learning rate to be altered as training progresses. This can be done in a number of different ways, including reducing the learning rate by a set amount after every q steps, using an exponential function to reduce the learning rate, or by implementing a polynomial, sigmoidal or inverse decay function.

Zeiler (2012) proposes ADADELTA, another gradient descent based technique, that improves upon the AdaGrad method (Duchi *et al.*, 2011) by removing the need to manually select a global learning rate as well as the need to continually update the learning rate throughout training. The technique accumulates the squared gradients for the last r iterations, which are then used to effectively reduce the learning rate. For the sake of efficiency, an exponentially decaying average of the squared gradients is taken. The average at time t is given as

$$L(g_t^2) = \mu L(g_{t-1}^2) + (1-\mu)g_t^2, \tag{3.7}$$

where μ is a constant used here similarly to momentum. Since the squared gradients are used in (3.7), the root mean squared is used to update the weights:

$$\text{RMS}(g_t) = \sqrt{L(g_t^2) + \epsilon}, \tag{3.8}$$

with ϵ a conditioning constant. The update to the weights also takes the previous updates into account:

$$v_t = \frac{\text{RMS}(v_{t-1})}{\text{RMS}(g_t)} g_t, \tag{3.9a}$$

$$w_{t+1} = w_t - v_t. \tag{3.9b}$$

Although neural networks have shown tremendous promise on a variety of image classification problems, there are some limitations that arise during training. First of all, multiple iterations are required in order for the system to reach a desirable accuracy. This might make the entire training process more time consuming. Furthermore, a large amount of training data is usually required for the network to generalize and to keep the network from overfitting. If these problems are not of much concern or can be addressed, these networks can be powerful.

3.3 Software

Software plays an important part in the training as well as execution of convolutional neural networks. It can allow a user to easily define a network and its layers, while also facilitating the training of a network and the execution of

a trained network. We decided to use Caffe to implement our networks, since it was readily available at the start of this study, is relatively straightforward to use and includes a Python wrapper.

In this section we mention some of the current most popular software choices that allow users to implement neural networks. A brief overview of Caffe, Theano and TensorFlow is presented.

3.3.1 Caffe

Caffe, created by Jia *et al.* (2014), is an open source framework that was developed in C++ with wrappers for both Matlab and Python. The software allows users to train and execute models on the GPU using CUDA, although CPU implementation is still permitted. There is also separation between the model definitions and implementation, allowing for simple model creation without low level design. The most popular layers as well as functions used in neural networks are also already implemented in Caffe and can be used by simply specifying the defining characteristics of the respective layers or functions.

Users are able to define a new model to train from scratch, with random parameter initialization, or to update an existing model using finetuning. A "model zoo", which consists of Caffe implementations of neural networks that are continually updated by researchers, is also made available.

3.3.2 Theano

Theano by Bergstra *et al.* (2011) is a Python library for symbolic variable execution that can be used to represent and train neural networks. Theano allows users to explicitly define their network using mathematical expressions that can then be evaluated on either the CPU or GPU quickly and accurately. Theano is also open source.

A network is built by first declaring symbolic variables and then constructing a graph with these variables. The most popular functions of neural networks are supported in Theano and used on the edges of the graph when the symbolic variables are connected. Theano seems to focus more on the lower level implementation of neural networks, allowing the user more control over exactly what the network does, at the expense of more high level functions that can be used without the user having to implement them.

3.3.3 TensorFlow

TensorFlow (Abadi *et al.*, 2015) is an open source Python API written on a C++ engine. Similar to Theano, it allows users to create computational graphs to represent a neural network. Multiple devices are supported by TensorFlow, and even distributed processing where a single network can be executed over several devices is supported. Neural networks are not the only models that

can be implemented by TensorFlow and its abstractions allow for other machine learning and numerical computation implementations. Most common functions associated with neural networks are implemented in TensorFlow.

3.4 Applications of neural networks

Neural networks have found many uses since they were first conceived. With improvements to hardware, these uses have themselves improved drastically and allowed neural networks to become more viable for a variety of tasks.

One of the most recent breakthroughs using neural networks comes in the form of AlphaGo, a deep neural network implementation by Silver *et al.* (2016) that plays the board game Go. The system uses neural networks to analyse the position of pieces on the board, and another network trained using both supervised learning from games played by experts, as well as reinforcement learning from games that the system played. AlphaGo was also the first artificial intelligence system able to defeat a professional human player in the game of Go, defeating the European Go champion by 5 games to 0. The system also later went on to defeat the world champion Go player by 4 games to 1.

Natural language processing is another area which has benefited from neural networks. Part of speech tagging, chunking and semantic roles labelling are some of the functions performed by Collobert and Weston (2008), and Collobert *et al.* (2011) using neural networks. Furthermore, Socher *et al.* (2011) extended the idea of parsing natural language sentences to images, where sentences or images are recursively broken down into distinctive parts.

Recurrent neural networks, which take the output of the previous pass through the network as an extra input parameter, have been implemented for data in which the sequence or timing of events is an added parameter. Graves *et al.* (2013) achieved low error rates when they trained a recurrent neural network for phoneme recognition.

Krizhevsky *et al.* (2012) famously won the ILSVRC-2012 competition for image classification by a large margin. The competition required contestants to classify input images into one of a possible 1,000 different categories. Using a convolutional neural network, the system achieved a top-5 error rate of 0.153. Face recognition and detection have also been explored using neural networks, as implemented by Lawrence *et al.* (1997) and Garcia and Delakis (2004).

As mentioned in Chapter 2, Jaderberg *et al.* (2014, 2016) implemented convolutional neural networks for both text detection and text recognition. In the text recognition stage, synthetic data was created to be used as a substitute for real world examples of text. Synthetic words with different fonts, colours, sizes and orientations were found to be a sufficient substitute for natural text for the task of recognition. State of the art accuracies were achieved when using this method. The work of LeCun *et al.* (1998) was probably one of the earliest neural network implementations for character recognition from images.

A convolutional neural network was implemented for the purpose of classifying handwritten digits. This implementation achieved state of the art accuracy when it was first released, and paved the way for research into text recognition with neural networks.

In the following chapter we describe our implementation of convolutional neural networks for the problem of text detection in natural images. Taking the various aspects explained in this chapter into account, we delve into the details regarding structure and training of the different networks we implemented.

Chapter 4

Implementation

The aim of this study is to find text in natural images. We implement a number of convolutional neural networks to classify smaller squares taken from an image (windows) as either text or not-text. Since the availability of datasets with ground truth annotation is quite limited, we investigate the feasibility of supplementing these datasets with synthetic data. We also consider different network structures, and initialize and train them using various combinations of natural and synthetic datasets. Finally, we discuss a possible method of implementing the trained networks to create saliency maps from full images.

4.1 Datasets

Two sources of data are used to train the neural networks. The first is a combination of different natural images that contain text, while the second consists of synthetically generated words added to a number of diverse background images. Convolutional neural networks require large amounts of labelled data to reach a good optimum during training, but annotated natural data is hard to come by and time consuming to produce. We created a synthetic dataset to compensate for the lack of natural images available, since synthetic text can be easily generated and automatically annotated. Multiple fonts, orientations, sizes and colours can also easily be incorporated.

An idea that we explore further is to use the large synthetic dataset to initially train a network, while the smaller natural dataset, which contains the type of image data that we ultimately want to classify, can be used to finetune the network.

4.1.1 Natural data

Sufficiently large datasets of images containing text can be difficult to come by. Furthermore, these datasets are often not annotated or only partially annotated, which may increase the time needed for preprocessing (before train-

ing can commence). As such, a relatively small set of natural images was collected. These images include the Google Street View Text (SVT) images collected by Wang and Belongie (2010), pictures of text in different settings (MSRA-TD500) compiled by Yao *et al.* (2012) and frames taken from the live stream on cnn.com. Since the ratio of text to background can be quite low in most images, single character images from the dataset composed by De Campos *et al.* (2009) (Chars74K) are also added to supplement the available text samples. Examples of images from all of these datasets are presented in Figure 4.1, while additional information about the datasets is provided in Table 4.1.

Dataset	Number of images	Number of text instances
SVT	350	2,061
MSRA-TD500	185	474
cnn.com	39	162
Chars74K	12,503	12,503

Table 4.1: The datasets and number of images that our natural training data consists of. The total number of areas annotated as text in each dataset is also presented.

The SVT dataset consists of 350 images taken from Google Street View. The main source of text in these images are from business signs and advertisements, although smaller and more obscure text can be found. The dataset was annotated by Wang and Belongie (2010), but we chose to add our own annotations, since many text instances were omitted from the original annotation files.

The MSRA-TD500 dataset consists of 500 images. These images contain different types of text, including photos of posters, advertisements, business signs and street signs. We only considered 185 out of the 500 images for our dataset, since many of the images contained characters from a different alphabet. We used the annotations provided by Yao *et al.* (2012) and simply converted them to our preferred format.

We pulled 39 frames from the live stream on cnn.com. These frames were added to include more variance between the text samples we use. Since these frames were pulled directly from the website, we annotated all instances of text by hand. Lastly, 12,503 single character images taken from the Chars74K dataset (De Campos *et al.*, 2009) were included to supplement the positive samples in our dataset.

Another dataset that we are aware of but chose to omit from our training data is the ICDAR dataset (Lucas *et al.*, 2003). This dataset consists of a training set of 250 images and a testing set of 249 images. The decision to exclude the ICDAR training set from our own was made in order to test how well our system would generalize on a completely unseen dataset.

(a) SVT image

(b) MSRA-TD500 image

(c) cnn.com image

(d) Chars74K image

Figure 4.1: Examples of images from the different natural datasets.

4.1.2 Synthetic data

Due to the limited size of available annotated natural data, we turned to creating our own data. Since this study aims to detect text, synthetically generated data seems to hold promise. This is due to the fact that text can be easily generated with relative diversity. We attempt to keep the underlying textural features of the synthetically generated data as close to the natural data as possible, by blending words of varying colours, fonts, sizes and orientations onto background images.

We devise a purposefully crude protocol for creating the synthetic dataset, so as not to influence the network into using predetermined features. Furthermore, the idea behind the synthetic data is to create a supplement to the natural data in such a way that the network would be able to generalize the features learnt during the training process when identifying text in natural images.

The process of generating the synthetic dataset is broken down into two parts, the first being to generate a word and the second to blend that word into the background image. An example of the entire process is shown in Figure 4.2.

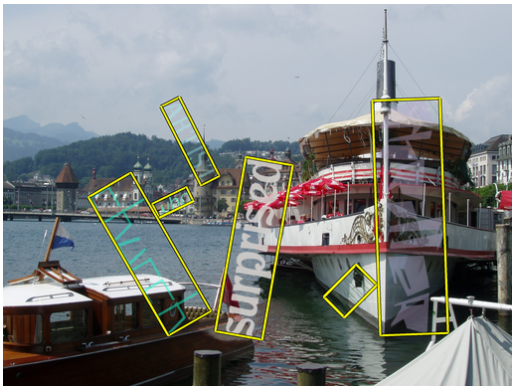
4.1.2.1 Word generation

A number of different words in a variety of fonts, colours, orientations, opacities and sizes are created to be blended onto background images. A word is randomly selected from a list of 10,000 English words, consisting of the most

(a) A background image.

(b) Generated words to be placed on the image.

(c) Words added to the image.



(d) Bounding boxes.

Figure 4.2: An example of the process of synthetically blending random words to a background image.

commonly used words on Google, and a corresponding font is randomly chosen from 723 unique fonts.

A font size is randomly generated based on the shortest edge of the background image to ensure that words are not dwarfed or overshadowed by the image. Next, each word is assigned a random colour and opacity, which ranges between 150 and 255 (with 0 being completely transparent and 255 completely opaque). Finally, a random angle ranging between $-90°$ and $90°$ is chosen as the orientation for the word.

After all parameters have been determined, an image of the word is created. The word image is created on a blank canvas to allow for an easier distinction between word and background when blending the word onto the background image. A word bounding box is also calculated, by taking the left-most, top-most, right-most and bottom-most pixels of the word before rotation, and rotating these points by the same angle around the centre point of the word. The four corners of the bounding box along with the word used are saved to be used when annotating the final image.

4.1.2.2 Blending words

A single image can have between one and ten different words added onto it. Ten words are randomly selected and generated as described in the previous section. Using the bounding boxes of the words, x- and y-offsets are randomly generated for each word, ensuring that the entire word would fit onto the image. The bounding box of the word is also checked against the bounding boxes of any words that have already been placed on the image, to prevent overlap between words. If there is no overlap the new word is added, otherwise it is simply discarded.

Randomly selecting a word's location and discarding a word if it does not fit, adds a measure of randomness to the number of words in the images. Since the words can be of different sizes and orientations, it also adds some variety between images and prevents them from being overcrowded.

Once a word has been added to an image, the word and the area around it is blurred. This is done to remove some of the sharp edges and colour contrast that can occur between a word and the background it is placed upon. An image of the word is used as a mask when blurring, to ensure that only the word and a small number of pixels around the word are blurred, instead of the entire image. The blurring is also implemented to simulate blurring of natural images which might distort the edges of characters.

The process of adding words to an image was chosen over the option of simply creating images of words with random noise as backgrounds. This decision was made both because the sliding window approach is used when trying to identify words, and to add a more natural variety to the backgrounds on which words can occur. Since the intersection between a sliding window and a word's bounding box often only contains partial letters when applied

to natural data, adding the synthetic words to natural backgrounds results in sliding windows that mimic this effect. Furthermore, this allows us to gather a wide variety of realistic negative samples along with positive samples.

4.1.3 Image windows and databases

The images containing text are subdivided into smaller, overlapping square images called windows. These windows are created to allow the system to classify them into either the text or not-text classes, rather than trying to directly identify all regions in an image that might contain text.

Windows are classified as text if the intersection between the ground truth bounding boxes of the text and the region the window is taken from covers at least 0.5 of the window area. Similarly, an image is classified as not-text if this intersection covers less than 0.05 of the window area. Examples of how windows are extracted and classified are shown in Figure 4.3. Any windows with intersections that fall between these two thresholds are discarded from the training and testing sets to allow clear differentiation between the two classes. A threshold of 0.5 rather than a higher number is chosen for the text class, not only to increase the number of positive samples in our dataset, but also since most sliding windows taken in the final system will only partially cover text instances. Even though some windows may only contain partial characters, the nature of a sliding window necessitates that they be classified as text, lest we miss instances of text. Only 0.1 of all windows classified as not-text are added to the datasets, to maintain balance between the number of positive and negative samples and avoid class bias during training.

Each image used for training, validation or testing is divided into smaller windows. The windows are classified using the Sutherland-Hodgman polygon clipping algorithm (Sutherland and Hodgman, 1974), which computes the polygon created by the intersection of two polygons. It operates by finding all line intersections between the two polygons and using the enclosed area as the new polygon. This process is repeated for all bounding boxes that intersect the window and the total area of text in the window is computed. This area is then used to assign a correct label to the window, or to discard it.

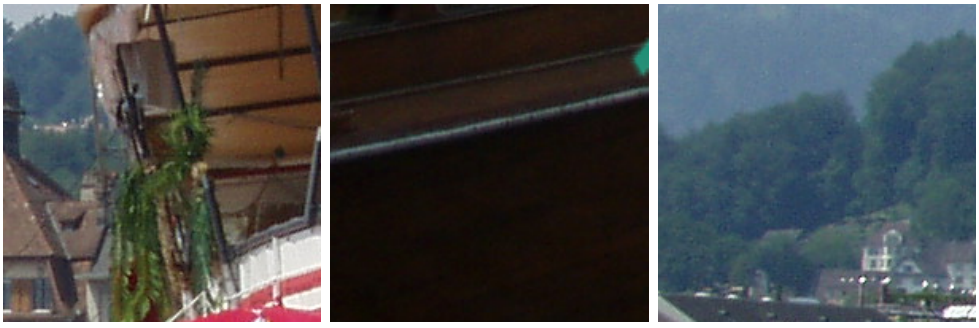
The resulting windows might differ in size, but are rescaled to 200×200 pixels. The images have to be rescaled in order to be used with the neural network, since each layer of the network takes a set number of variables which, in the case of the first layer, are the pixels of the input image window. The image channels are also swapped from the traditional RGB to BGR, while the image array is transposed from the normal height $\times$ width $\times$ channels, to channels $\times$ height $\times$ width. This is all done to account for the conventions followed by Caffe.

Finally, the images are saved to a database to allow for quicker access by the networks. Python's LMDB database structure is used since it is fully

(a) An image containing synthetically generated words.

(b) Examples of windows taken from the image.

(c) Windows labelled as positive samples (text).



(d) Windows labelled as negative samples (not-text).

(e) Windows ignored because they contain between 0.5 and 0.05 text.

Figure 4.3: An example of how windows are taken from an image, as well as examples of windows labelled as each of the two classes and windows that are discarded.

supported by Caffe, is readily available and does not require any license when used for research purposes.

In total, seven datasets are created. The first two datasets are created using the synthetically generated data, while the final five consist solely of windows taken from the natural data. The first dataset, `synth_train`, consists of 1,090,709 windows taken from the synthetic data, with 416,195 windows labelled as text and 674,514 windows labelled as not-text. The second dataset created from the synthetic data, `synth_val`, consists of 307,721 windows, of which 115,782 are labelled as text and the remaining 191,939 as not-text. As can be inferred from the names of the datasets, the `synth_train` dataset is intended to train neural networks, while the `synth_val` dataset is intended to serve as a validation set during training.

Three datasets are created from the natural data for the purpose of training neural networks. These datasets can be differentiated from one another by the number of images used for each as well as the number of windows that each contains. The idea behind these multiple datasets is to simulate different amounts of available training data.

The first and smallest dataset, `natural_s`, is created from 200 natural images, which were divided into 15,837 windows, 8,223 labelled as text and 7,613 as not-text. Next, `natural_m` is created from 320 images, from which 25,979 windows, 13,758 text and 12,221 not-text, were created. The last and largest natural training dataset was created by dividing 373 natural images into windows, as well as adding 12,503 natural single character images. This dataset contains 57,714 windows, with 28,880 windows labelled as text and 28,834 windows labelled as not-text. Note that the images used for the small dataset are a subset of the images used for the medium dataset, which in turn is a subset of the images used for the large dataset.

Finally, a validation set, `natural_val`, as well as a hold out test set, `natural_test`, are created from 86 and 115 natural images respectively, which

Dataset	# images	# windows	# positives	# negatives
`synth_train`	3,634	1,090,709	416,195	674,514
`synth_val`	1,023	307,721	115,782	191,939
`natural_s`	200	15,837	8,223	7,613
`natural_m`	320	25,979	13,758	12,221
`natural_l`	373 + 12,503	57,714	28,880	28,834
`natural_val`	86	7,640	3,750	3,890
`natural_test`	115	10,695	5,488	5,207

Table 4.2: Sizes and details of the different datasets. The positives refer to windows labelled as text, while negatives refer to windows labelled as not-text. The additional 12,503 images in the largest natural dataset are single character images.

are not used in any of the training sets. The validation set consists of 7,640 windows with 3,750 examples of text and 3,890 examples of not-text. The hold out test set is larger, containing 10,695 windows of which 5,488 are labelled as text and 5,207 as not-text. A complete breakdown of the different datasets is presented in Table 4.2.

4.2 Neural networks

Convolutional neural networks are trained to classify the image windows mentioned above into either the text or not-text categories, by assigning a probability to each category. Using different structures, parameter initializations and data, 14 unique networks are created and trained.

This section explores how the different neural networks are created and trained. We first take a look at the different structures employed, after which we discuss how the networks can be differentiated based on initialization and the training procedures followed. An overview of the different networks is presented in Tables 4.3 and 4.4 (in Section 4.2.2).

4.2.1 Structure

Determining the best structure to use for a given task can be intricate and often relies on trial and error. We decided to implement two structures that had previously been used successfully on similar tasks. The first structure, LeNet, consists of two convolutional layers followed by a single fully connected layer and a softmax layer, and was first implemented by LeCun *et al.* (1998) for the task of document text recognition. The second structure, ImageNet, is much larger than the first, consisting of five convolutional layers followed by two fully connected layers and a softmax layer, as implemented by Krizhevsky *et al.* (2012) for the task of object recognition.

The first layer of the LeNet network consists of 20 filters of size 5×5, with a step size of 1 pixel. Max pooling (a form of sub-sampling) is applied on a 2×2 pixel basis to the output of the filters, to produce a matrix of size 98×98. The next convolutional layer contains 50 filters, once again of size 5×5 with a step size of 1. Max pooling is once again applied on a 2×2 pixel basis, passing 50 matrices of size 47×47 on to the fully connected layers. The first fully connected layer consists of 500 neurons, first taking the inner product of the input with their weights, after which a rectified linear function is applied. Finally, the result of the 500 neurons are passed to 2 neurons in the softmax layer which compute the inner product and present the output of the network in the form of class probabilities. The structure of the LeNet network is shown in Figure 4.4.

The structure of the ImageNet network is a bit more complex, and shown in Figure 4.5. The first convolutional layer contains 96 filters of size 11×11,

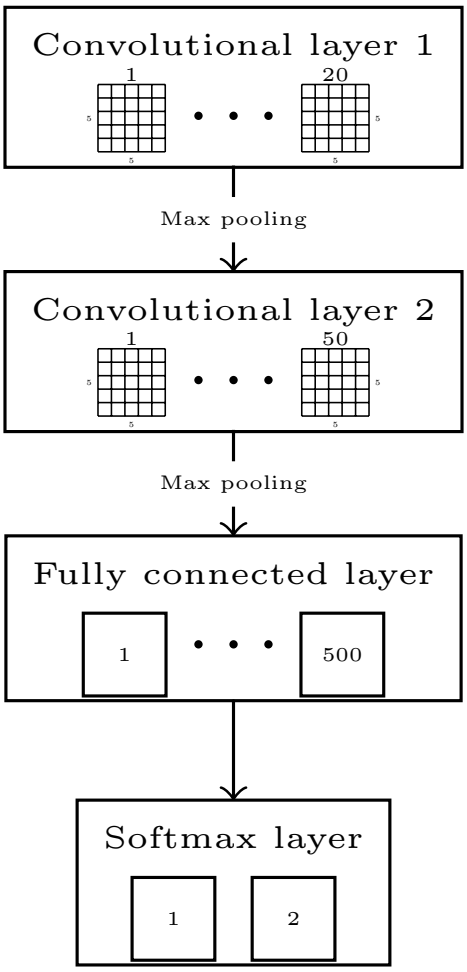


Figure 4.4: The structure of the LeNet network. Max pooling is applied in a 2×2 area with a stride of 2 in both convolutional layers, while the fully connected layer implements a rectified linear function.

with a step size of 4 pixels. The first layer already has a built in nonlinearity in the form of a rectified linear function, followed by overlapping max pooling in a 3×3 area, with a step size of 2 pixels. The results are normalized, which aids with generalization, before being passed on to the next layer.

The next layer follows more or less the same trend, with 256 filters of size 5×5 with a step size of 1 pixel, a rectified linear function, overlapping max pooling in a 3×3 area, with a step size of 2 pixels and normalization. The third layer, with 384 filters of size 3×3 and a step size of 1 pixel has neither a pooling layer nor a normalization layer, and applies only the rectified linear function. The fourth layer follows suit with the third, having the same number and sizes of filters. The fifth and final convolutional layer once again introduces overlapping max pooling in a 3×3 area with a step size of 2. This layer reduces the number of filters from the previous one, having 256 filters of size 3×3 and

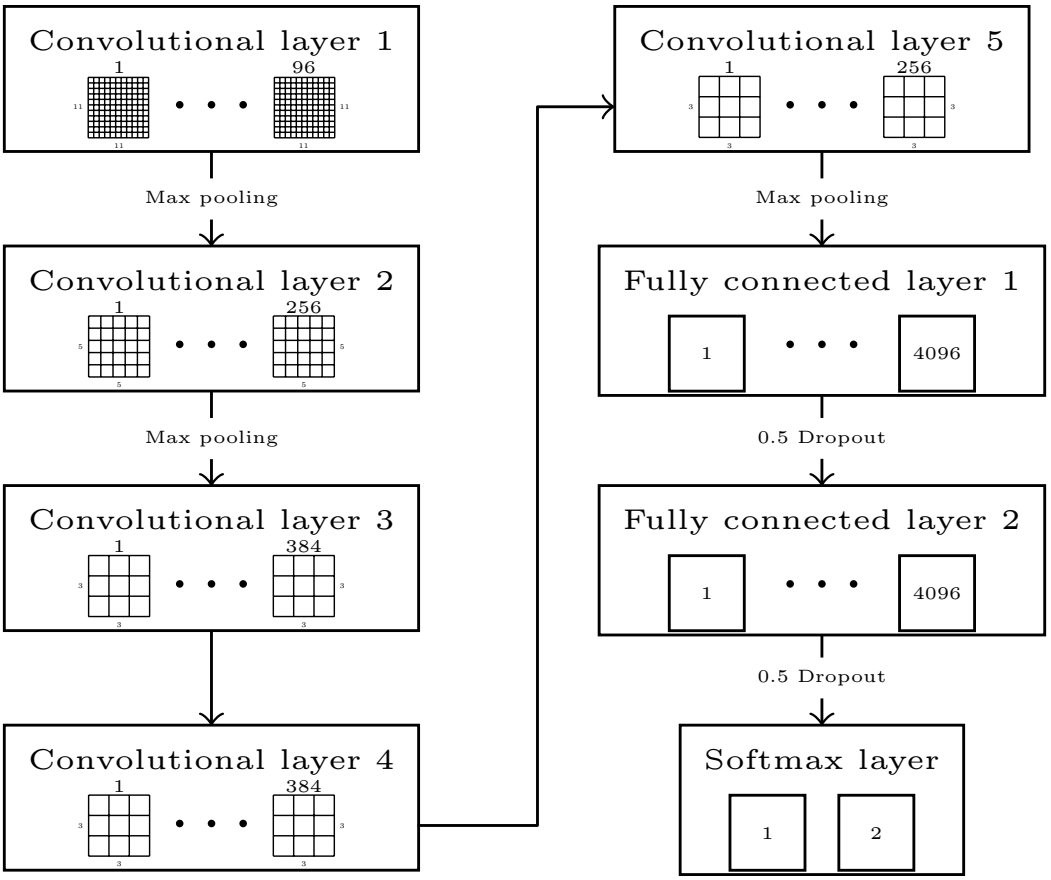


Figure 4.5: The structure of the ImageNet network. Max pooling is applied in a 3×3 area with a stride of 2, while normalization is applied in the first two convolutional layers. All of the layers, except for the softmax layer, also employ a rectified linear function.

a step size of 1 pixel.

The fully connected layers introduce dropout into the network. The first fully connected layer contains 4,096 neurons that first take the inner product of the input with their weights and then apply the rectified linear function. Dropout is introduced by setting the output of a neuron to zero with probability 0.5, reducing overfitting on the training data. The second fully connected layer is the same as the first, while the softmax layer simply has two neurons that compute the inner product of their inputs, and presents the output of the network in the form of class probabilities.

4.2.2 Network initialization, variables and training

Networks with different structures are trained separately using different initializations, training variables and adaptive learning rate functions. Initial tests were run, experimenting with different learning rates and functions, to weigh

up which would be best for each individual scenario. Inconsistencies between the training variables used for networks with the same dataset and structure are, for example, due to their initializations, where using the same variables lead to convergence for one network and divergence for another. A summary of the initialization variables for all networks with a LeNet structure is presented in Table 4.3, while the variables for networks with an ImageNet structure are presented in Table 4.4.

For networks that use the LeNet structure, filter weights are initialized using the Xavier algorithm (Glorot and Bengio, 2010) while bias weights are simply initialized to zero. The Xavier algorithm attempts to initialize the network weights in such a way as to prevent the input from growing too large or small as it is passed through the network. This is accomplished by drawing the weights from a zero mean Gaussian distribution with variance $\text{var}(W_i) = \frac{2}{n_{in}+n_{out}}$ where

LeNet			
Network	Initialization	Initial LR	Training data
A	Xavier	10^{-5}	`synth_train`
B	Xavier	10^{-5}	`natural_s`
C	Xavier	10^{-5}	`natural_m`
D	Xavier	10^{-5}	`natural_l`
E	Network A	10^{-5}	`natural_s`
F	Network A	10^{-5}	`natural_m`
G	Network A	10^{-5}	`natural_l`

Table 4.3: Initialization, initial learning rates and training data for networks using the LeNet structure. All networks followed an inverse decay learning rate policy and were trained in 10,000 iterations.

ImageNet			
Network	Initialization	Initial LR	Training data
H	Gaussian	10^{-4}	`synth_train`
I	Gaussian	10^{-5}	`natural_s`
J	Gaussian	10^{-5}	`natural_m`
K	Gaussian	10^{-5}	`natural_l`
L	Network H	10^{-6}	`natural_s`
M	Network H	10^{-6}	`natural_m`
N	Network H	10^{-6}	`natural_l`

Table 4.4: Initialization, initial learning rates and training data for networks using the ImageNet structure. All networks, except for Network H, followed an inverse decay learning rate policy and were trained in 50,000 iterations. Network H followed the STEP learning rate policy and was trained in 225,000 iterations.

n_{in} and n_{out} are the number of neurons in the previous and following layers of the network respectively. This is done to keep the variances between layers similar for both forward passes and back propagation. Although the Xavier algorithm was not implemented in the initial training of the LeNet network (LeCun *et al.*, 1998), it was later added to improve convergence of the network.

Networks that employ the ImageNet structure sample values from a Gaussian distribution with standard deviation 0.01 for the weights of the convolutional layers, and a Gaussian distribution with standard deviation 0.005 for the weights of the fully connected layers. Bias weights are once again initialized to predetermined values, ranging between 0 and 1 depending on the layer. These distributions and values are chosen to coincide with those used by Krizhevsky *et al.* (2012), who spent some effort on optimization.

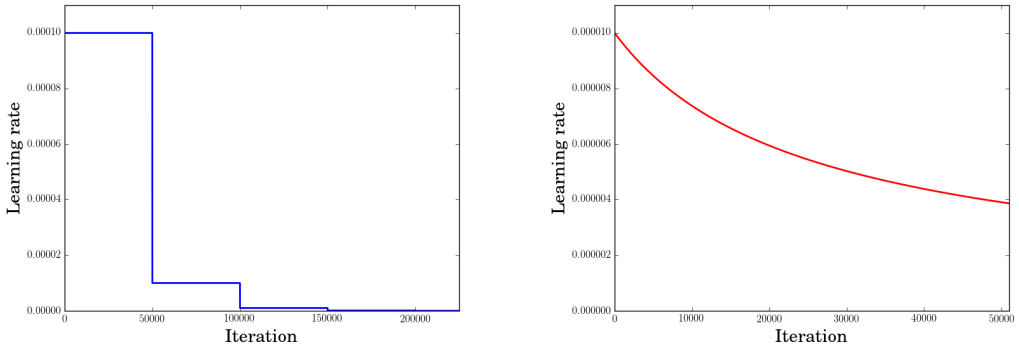
Finally, more networks that employ both the LeNet and ImageNet structures are initialized to the starting weights of networks initialized as specified above and trained on synthetic data. The idea behind this approach is that, although the synthetic data might not fully resemble the natural data, enough similarities persist between the two datasets that features detected by a network trained on the synthetic data will still be useful when detecting text in natural images. As a result, the weights from these networks, which have been trained on much larger datasets, might allow networks using them as a starting point to end up in better optima than when randomly initialized.

The learning rate and its modification policy are arguably the most important aspects of training a network, and can be quite difficult to optimize. An efficient learning rate allows the network to improve upon its previous prediction by adjusting the weights of the network in such a way that the final output is closer to the expected output. If the learning rate is too small, the network could easily get stuck in a local optimum, while learning rates that are too large may lead to divergence without a solution being found.

Since determining a good learning rate is a difficult task, different experiments were performed with varying learning rates to test which would be optimal for each network. By varying the learning rate and tracking the training and validation accuracies of the network throughout the training process, it becomes somewhat easier to see which learning rates are too large or too small for each network.

After multiple tests, the learning rate for all the networks with a LeNet structure were set to 10^{-5}. The network with an ImageNet structure that was trained on the synthetic data had the largest initial learning rate, with 10^{-4}. The reason that a larger initial learning rate was needed could be due to the fact that the ImageNet structure is much larger than the LeNet structure, with more variables that might need larger changes to be able to move out of local optima. Another interesting result was that the networks with ImageNet structure that were initialized to the pre-trained network, worked better with a smaller learning rate of 10^{-6}, compared to their randomly initialized counterparts that required an initial learning rate of 10^{-5}.

A learning rate policy is also applied for each network to decrease the learning rate as training progresses. Two learning rate adaptation policies were implemented in the networks, namely the STEP and inverse decay policies. The inverse decay policy is a continuous function that decreases the learning rate based on some parameters, while the STEP policy discretely reduces the learning rate after every n iterations. Examples of both these policies are presented in Figure 4.6.



(a) STEP policy.

(b) Inverse decay policy.

Figure 4.6: The learning rate adaptation policies implemented.

Inverse decay was applied to the learning rates of all networks except the network with ImageNet structure trained on synthetic data. The inverse decay policy allows for a smoother decrease in the learning rate, and is better suited for networks where convergence takes place quicker. This is often the case for smaller networks or networks that are trained on smaller datasets. The learning rate at iteration i is given by

$$\alpha_i = \frac{\alpha_0}{(1+\gamma i)^p}, \tag{4.1}$$

where α_0 is the initial learning rate and γ and p are variables to determine the rate of change. For all the networks using the LeNet structure, we chose $\gamma = 0.0001$ and $p = 0.75$, while the networks with the ImageNet structure used $\gamma = 0.00005$ and $p = 0.75$. These variables were once again chosen through experimentation on the training and validation sets.

The only network that employs the STEP policy is the network with the ImageNet structure that was trained on the synthetic dataset. The STEP policy operates by reducing the learning rate by a factor $\frac{1}{\gamma}$ every s iterations. The learning rate at iteration i is given as

$$\alpha_i = \alpha_0 \gamma^{\left\lfloor \frac{i}{s} \right\rfloor}. \tag{4.2}$$

For the network using the STEP policy, we chose $\gamma = 10^{-1}$ and $s = 50{,}000$ which effectively decreases the learning rate by a factor of 10 every 50,000

iterations. This method is preferable for training a large network on a large dataset, since it allows more time for larger changes in the network weights when compared to the inverse decay policy. Furthermore, the network is able to make both large and small changes based on images from the entire dataset, rather than a smaller subset, since the network is trained on multiple batches for each learning rate.

The weights are updated after each batch of images is passed through the network, using the learning rate as a scaling mechanism to determine the size of the weight update. The weights of the bias inputs to each neuron are updated using a learning rate equal to twice the normal learning rate. Although the reasoning behind this is not clear, we follow the implementation of Krizhevsky *et al.* (2012) since it appears to aid convergence. The stochastic gradient descent function used for updating a single weight vector, described in Section 3.2, is used for training:

$$v_{t+1} = \mu v_t - \alpha \nabla E(w_t), \tag{4.3a}$$

$$w_{t+1} = w_t + v_{t+1} - \lambda \alpha w_t, \tag{4.3b}$$

where α is the learning rate, μ the momentum, λ the weight decay, v_t the previous weight update, w_t the current weight of the neuron and $\nabla E(w_t)$ the derivative of the loss function. This effectively modifies the weight vector in the direction of greatest descent for the loss function, while simultaneously strengthening changes that follow the common trend and penalizing weights that become too large.

A momentum value of 0.9 is used for all of the networks. This value allows the network to take previous weight updates into account when training. Adding momentum during training helps the network to escape local optima, as well as speeding up the training process. Since the previous update to a weight is simply scaled and added to the updated weight value, all previous updates are effectively taken into account. The impact of the previous updates, however, is diminished the further back the update occurred.

Momentum speeds up training by enforcing sequential changes in the same direction, while at the same time inhibiting changes in opposite directions. What this comes down to, is that if multiple batches would benefit from either increasing or decreasing a certain weight, all updates that go along with this trend are strengthened, while updates that go against it are weakened. This prevents outliers from influencing the updates too much, thus leading to faster convergence.

When the network gets stuck in a local optimum, it usually means that the derivative of the loss function is so small that the update to the weight is not significant enough to improve the prediction. By implementing momentum, this problem is reduced since large previous updates, combined with these smaller, weaker updates, allow the network to make larger, sensible changes to the weights.

The last training variable, weight decay, is used for regularization of the weights in the network. Weight decay causes a weight to reduce in proportion to its size, by subtracting a scaled version of the variable from the updated weight vector. Applying weight decay punishes larger weights more than their smaller counterparts, helping prevent a weight from becoming too large and overshadowing the other weights. A weight decay of 0.0005 is applied to all networks.

Batch training is implemented when training the network. This means that the network variables are only updated once a batch of images has been passed through the system, by taking the average loss over all images in the batch to update the weights. Using batch training, instead of the alternative of updating the network after each image, helps to remove some noise and better generalizes on smaller datasets. Since smaller datasets of natural data are used, we decided to implement batch training, with a batch size of 128 for all networks trained on natural data and 256 for networks trained on synthetic data. A larger batch size was used for the synthetic data, since the dataset is much larger. We note that the batch size is also limited by the memory of the video card used to train the networks.

Finally, the networks were trained until a predetermined number of batches had been passed through. The number of batches, or iterations, were once again chosen by testing the network on the validation set using different numbers of iterations. Since the size of the learning rate is decreased as the network is trained, at a certain number of iterations the learning rate will be too small to make significant changes to the weight variables. All networks with the LeNet structure were trained in 10,000 iterations, while the networks with ImageNet structure, excluding the one trained on the synthetic dataset, were trained in 50,000 iterations. The network with ImageNet structure and trained on the synthetic dataset was trained using 225,000 iterations.

4.3 Text detection in full images

The networks discussed in the previous section are trained to classify small square images into either the text or not-text categories, and we employ a sliding window approach to scan an entire image. The output of this part of our approach is a saliency map with the same size as the original image.

Initially, the full image is subdivided into smaller windows that are each passed through a trained network. Each window covers an $n \times n$ area of pixels, and the corresponding pixels in the saliency map are updated to reflect the probabilities returned by the network. To account for a wider range of possible text sizes, image pyramids can also be employed to generate multiple saliency maps at different scales, which are combined to produce a final saliency map.

4.3.1 Windowing

Our networks are trained to classify local image regions (windows) as containing text or not. As such, an input image has to be subdivided into smaller windows that can each be classified. The classifications for all windows have to be combined to form a saliency map that identifies regions of text in the original image.

We make the assumption that the text in the image is roughly proportional to the size of the image. Thus we choose the window size such that ten windows would fit the width of the image. This choice of window size coincides with that of the training sets. Since the network requires all input to be of a fixed size, the windows have to be resized to be 200×200 pixels before being passed through the network.

For the sake of simplicity and to improve classification times, a ratio preserving resampling operation is applied to the input image to fit ten 200×200 pixel windows. Some preprocessing operations required by Caffe, like channel swapping, are also performed on the larger image to prevent unnecessary repetitions.

When extracting windows, we also decided to introduce overlap between consecutive windows. Introducing overlap gives the network a chance to correct errors that might result from a poorly chosen window, and effectively increases the resolution in the resulting saliency map. We choose a step size of 100 pixels, so that two consecutive windows will share half of their pixels. Although smaller step sizes result in smoother and sometimes more accurate saliency maps, they do lead to an increase in execution time.

The windows are passed through the network one at a time. The probability of text output for each window is saved for the pixels that correspond to that window. Due to the overlap caused by the smaller step size, different probabilities might be assigned to a single pixel. To determine the probability of a pixel being text, the average probabilities of all windows that include that pixel are taken. This simple linear combination of probabilities turns out to be quite effective, and is similar to the strategy of Lienhart and Wernicke (2002). We may also decide to classify a pixel as text if the average probability of text is at least 0.5.

4.3.2 Image pyramids

Although our training data contains text of various sizes, we use a constant size when selecting windows from the larger training images. As such, larger text sizes might be incorrectly classified by the network. These text occurrences should be identifiable if larger windows are used for detection.

Image pyramids, which consist of multiple saliency maps generated at different scales, are implemented to account for a wider range of possible text sizes, as well as to improve the accuracy of the system. Although the neural

networks are trained with text of different sizes, the windows used for classifying full images can be too small to distinguish larger text from background structures. Lienhart and Wernicke (2002) assert that combining saliency maps generated at different scales improves the final accuracy, since we expect text to be detected at multiple scales.

We once again start off with a window size that is one tenth of the input image width. Once a full saliency map has been computed for an image, we increase the window size by a factor of 1.25 in both dimensions. We choose this factor for two reasons. Firstly, there exists some variance in the text sizes present in the training set, which would make a smaller increase redundant. Secondly, the smaller the increase to the window size, the more saliency maps have to be generated, increasing the computational time of the system. The process of increasing the window size is repeated until windows no longer fit into the original image. For most of the images we considered, 8 saliency maps are created before the windows become too large. The execution time also decreases with each new saliency map, since fewer windows fit into the original image. The saliency maps are generated following the same steps presented in the previous section.

After the different saliency maps in an image pyramid have been created, they have to be combined into a single saliency map. Different combinations of saliency maps can also be used to create the final saliency map and can be done in a number of ways. We opted to apply majority voting, where a pixel is only classified as text if more than half of the saliency maps classify it as text. This simple process is displayed in Figure 4.7 and shows how multiple classifiers can be combined to improve the accuracy of any individual classifier.

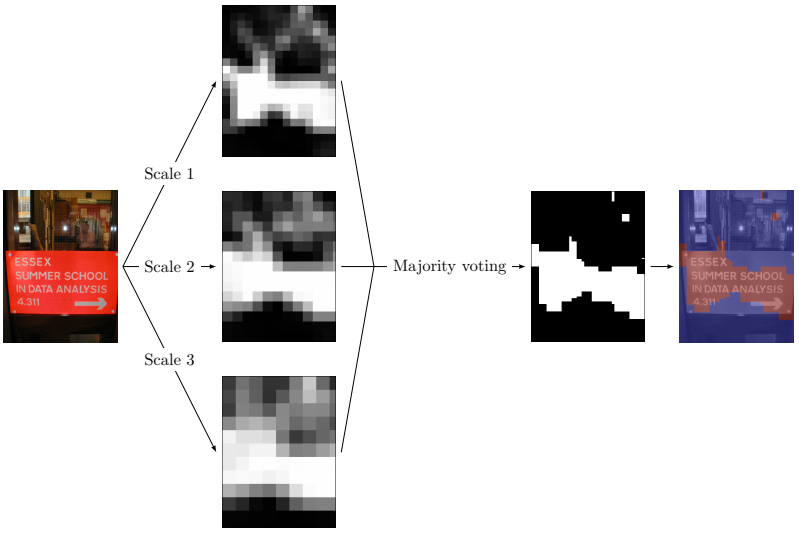


Figure 4.7: A representation of how image pyramids are combined using majority voting. This example uses an image pyramid generated at three different scales.

This chapter focussed on the implementation details of our system for detecting text in images. In the next chapter we present and discuss results obtained on our testing data for the various networks and also the full image classification system, and attempt to compare these results to existing text detection methods.

Chapter 5

Results

In this chapter we present our results obtained when implementing the various neural networks described in the previous chapter, as well as results obtained when applying the networks to the task of detecting text in full images. We also compare the performance of our system with that of different existing approaches on the ICDAR dataset (Lucas *et al.*, 2003) which our networks have not seen before.

Our networks were created, trained and executed using Python and Caffe, under the Ubuntu 14.04 operating system. An Nvidia GTX Titan X graphics card with 12 GB of memory was used for the training and execution of the networks. The system further has 16 GB of memory and an Intel Core i7-2700k 3.50 GHz processor.

5.1 Classification of image windows

In this section we discuss the results obtained by the different neural networks on the task of classifying individual image windows as text or not-text. As explained in Chapter 4, different datasets are used for training, including synthetically generated and natural datasets. Two network structures are also implemented, namely the LeNet and ImageNet structures. The weights of the network are initialized either randomly or to the weights of a previously trained network.

When referring to the accuracy achieved by a network, the combined true-positive and true-negative rate is meant. Recall measures the tendency of the network to detect text, while precision measures the likelihood of a text classification being correct. An image window is classified to a class if the network assigns the highest probability to that class. Since our problem contains two classes, this effectively means a window is assigned to a class if the network returns a probability of at least 0.5 for that class. Our results obtained on the 14 different convolutional neural networks are summarized in Table 5.1. In the rest of this section we proceed with a detailed analysis of the results.

Network	Structure	Initialization	Training set	Train acc.	Val acc.	Test acc.	Recall	Precision
A	LeNet	Random	`synth_train`	0.817	0.814	0.722	0.535	0.876
B	LeNet	Random	`natural_s`	0.996	0.748	0.750	0.653	0.822
C	LeNet	Random	`natural_m`	0.989	0.794	0.788	0.705	0.857
D	LeNet	Random	`natural_l`	0.971	0.789	0.767	0.637	0.876
E	LeNet	Network A	`natural_s`	0.995	0.776	0.777	0.690	0.848
F	LeNet	Network A	`natural_m`	0.975	0.803	0.803	0.706	0.887
G	LeNet	Network A	`natural_l`	0.954	0.811	0.810	0.716	0.893
H	ImageNet	Random	`synth_train`	0.948	0.942	0.663	0.385	0.904
I	ImageNet	Random	`natural_s`	0.926	0.825	0.792	0.709	0.860
J	ImageNet	Random	`natural_m`	0.886	0.839	0.812	0.766	0.859
K	ImageNet	Random	`natural_l`	0.884	0.833	0.843	0.800	0.884
L	ImageNet	Network H	`natural_s`	0.962	0.825	0.817	0.694	0.931
M	ImageNet	Network H	`natural_m`	0.948	0.856	0.852	0.762	0.938
N	ImageNet	Network H	`natural_l`	0.929	0.857	0.870	0.816	0.922

Table 5.1: Comparison between network structures, initialization and accuracy on the training, validation and testing datasets, as well as the precision and recall obtained on the test set. The datasets `natural_val` and `natural_test` are used as a validation set and test set, respectively, for all 14 networks.

5.1.1 Data

As discussed in Section 4.1, different datasets are used in the training and testing of the networks. These datasets are divided into sets created from natural images and synthetically generated images. For training and measuring accuracy in testing, the larger images are subdivided into smaller windows which are labelled as either text or not-text. A window is labelled as text if at least 0.5 of its pixels are contained within the ground truth bounding boxes identifying text in the image, while windows with less than 0.05 of their pixels falling within these bounding boxes are labelled as not-text. Details of the different datasets are presented in Table 4.2.

The `natural_test` set, which our networks have not seen before, is used to measure the accuracy of all the networks. As seen in Table 5.1 training with natural data leads to a better test accuracy than with synthetic data, regardless of the size of the natural dataset. It is also evident that networks trained on `natural_l` and `natural_m` perform better than those trained on `natural_s`, indicating that larger training sets lead to increased performance. It is worth noting, however, that the accuracies obtained with `natural_l` and `natural_m` are similar, and that `natural_m` even performs slightly better in the case of Networks C and D. This could be because a much larger dataset is needed to see any significant improvement, or that the single character images that were added to `natural_l` were too similar to the samples already in the dataset. Apart from those single character images, the two datasets do share a large number of source images.

5.1.2 Structure

Two different network structures were considered, namely the LeNet (LeCun *et al.*, 1998) and ImageNet (Krizhevsky *et al.*, 2012) structures. The LeNet structure consists of two convolutional layers and two fully connected layers. ImageNet is much more complex, consisting of five convolutional layers and three fully connected layers.

The ImageNet networks outperform their LeNet counterparts in all cases where the same training data is used. This is most likely due to the large difference in the number of filters and variables between the two structures. A more reliable function that better fits the problem space might be achievable with a more complex structure.

5.1.3 Initialization

Network initialization refers to how the starting values for the weights in the network are chosen. We considered both random initialization (and proceeded to train from scratch) and initializing the weights to those of another, previously trained network with the same structure (and proceeded with finetuning).

Network A, which has a LeNet structure and was trained on synthetic data, as well as Network H, which has an ImageNet structure and was trained on synthetic data, were used as a starting point for networks finetuned on natural data.

The networks initialized to the variables of the pre-trained networks, perform better than their randomly initialized counterparts, regardless of the training set or structure of the networks. This enforces the idea that first training on a large synthetic dataset allows a network trained on a smaller dataset to better identify and classify relevant features. It is likely that the smaller natural datasets do not contain enough data and variation to allow the networks to efficiently identify relevant features, while the networks trained on the larger synthetic datasets seem to be able to accomplish this task. The features extracted by the synthetically trained networks remain relevant for natural data, and thus using these networks as a starting point for finetuning improves overall performance.

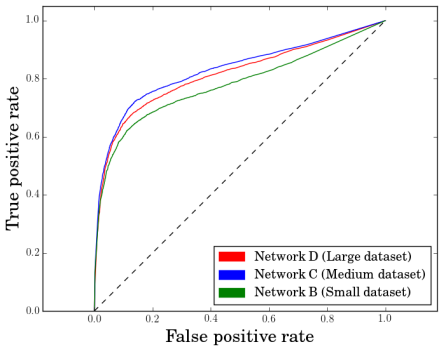
5.1.4 Accuracy

An image window is classified into the text class if the network returns a probability of 0.5 or higher that the window contains text. In Table 5.1 we list the accuracies achieved by each network on its training and validation set during training, as well as the accuracies achieved after training on the test set, `natural_test`.

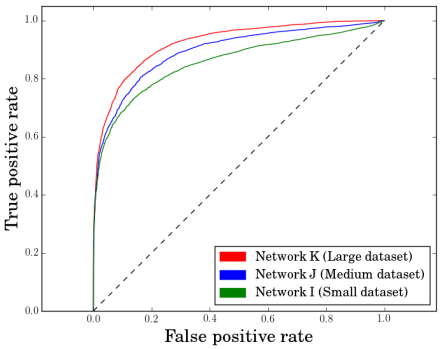
The test accuracy of networks trained using only the synthetic dataset is lower than when the networks are trained using a natural dataset, regardless of network structure or initialization. This might be expected, since even though we attempted to create synthetic images that capture the underlying textural features of text in natural images, there is a noticeable difference between the synthetic and natural data.

All of the networks trained on natural data achieved training accuracies higher than validation and test accuracies. This indicates that some overfitting occurred, possibly due to the training sets being too small. The difference between training accuracy and validation accuracy does decrease as the training sets become larger. This indicates that larger training sets may be required for the networks to generalize properly and avoid overfitting. Ideally, a network would have more or less the same accuracy over its training, validation and test sets. This could probably be obtained by using a large enough dataset, as is evidenced by the networks solely trained on the synthetic dataset, where the difference between the training and validation accuracies are comparatively small.

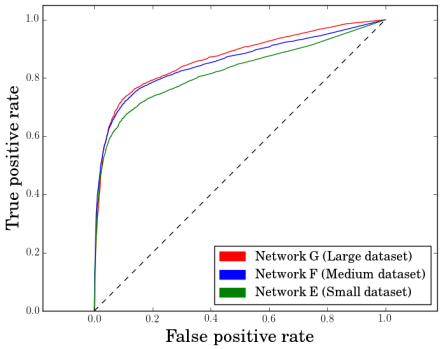
We present ROC curves of all the trained networks, as well as a comparison between the best performing randomly initialized, finetuned and synthetically trained networks in Figure 5.1. These graphs plot the true positive rate against the false positive rate of a network, as achieved on the test set, parameterized



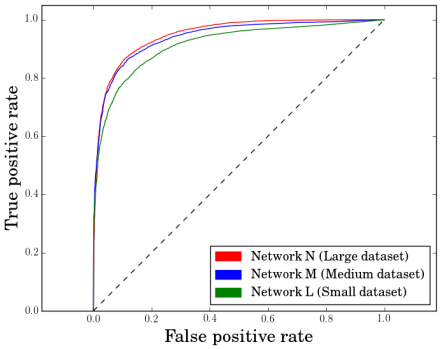
(a) Randomly initialized LeNet networks.



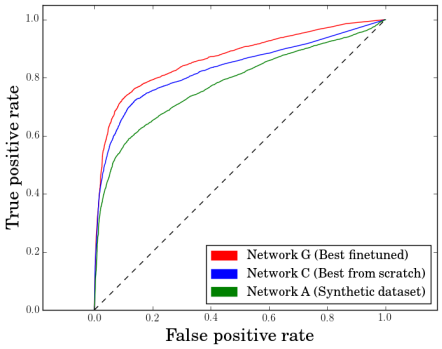
(b) Randomly initialized ImageNet networks.



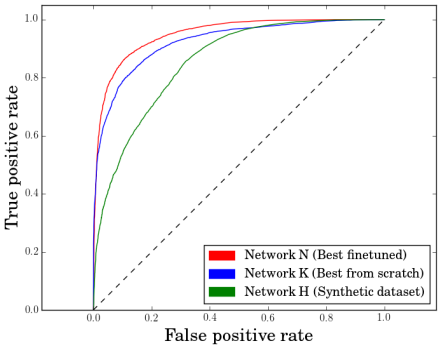
(c) Finetuned LeNet networks.



(d) Finetuned ImageNet networks.



(e) Comparison between best performing randomly initialized, finetuned and synthetically trained LeNet networks.



(f) Comparison between best performing randomly initialized, finetuned and synthetically trained ImageNet networks.

Figure 5.1: ROC curves depicting how all the networks perform on the test set, parameterized by the threshold that converts a returned class probability into positive (text) or negative (not-text).

by the threshold that turns class probabilities into classifications. They clearly show that the ImageNet structure outperforms the LeNet structure, and that the finetuned networks outperform their randomly initialized counterparts.

5.1.5 Recall and precision

Recall and precision are also given in Table 5.1 for each network, to give a better idea of each network's ability to identify text. This is useful since we want the networks to detect text accurately, and might be less concerned by their ability to identify windows in the not-text class.

For networks with an ImageNet structure, it appears as if recall is not influenced much by initialization, although precision seems to increase when a network is initialized using the weights of a network trained on synthetic data. The same is true for networks with the LeNet structure, although recall in this case does seem to increase for most networks.

5.1.6 Training times

Since convolutional neural networks typically contain millions of parameters and often require a large number of iterations before reaching a good optimum, training can be a time consuming process. Once a network is trained, however, classification happens much faster. It is often difficult to determine the number of iterations needed for a good optimum to be reached, since the size of the training set and variables that influence training all have an effect on that number. The training time depends mostly on the number of iterations, although network structure may also have an effect.

We simply present the training times for the two network structures trained on both natural data and synthetic data in Table 5.2. Note that the LeNet network trained on synthetic data is much slower than networks with the same structure trained on natural data, even though they used the same number of training iterations. This is due to the fact that the synthetic datasets are much

Structure	Training set	Iterations	Time
LeNet	Synthetic	10,000	1 hour 50 min
LeNet	Natural	10,000	30 min
ImageNet	Synthetic	225,000	42 hours 50 min
ImageNet	Natural	50,000	1 hour 50 min

Table 5.2: A comparison between the training times of networks with different structures, training datasets and number of iterations. Network A has a LeNet structure and synthetic training data, while Networks B, C, D, E, F and G have a LeNet structure and natural training data. Network H has an ImageNet structure and synthetic training data, while Networks I, J, K, L, M and N have an ImageNet structure and natural training data.

larger, which results in longer tests on the validation set during training, as well as the datasets taking longer to load into memory. If these factors were removed, the training times for all networks with a LeNet structure and a fixed number of iterations would be more or less the same.

5.1.7 Filters

Each convolutional layer in a network consists of multiple filters that are applied to the input of the layer. The filter weights are either randomly initialized or initialized to the weights of a pre-trained network, and adjusted during training. When comparing some of the filters visually, interesting features can be noted. The learned filters in the first convolutional layer of three networks with the ImageNet structure are visualized in Figure 5.2.

The filters as well as outputs for the first convolutional layer in both Network H and Network N appear to be similar. Some structures, reminiscent of Gabor filters, are also present in both of these two filter sets. It is to be expected that the filters would be similar, since the learning rate in finetuning Network N was quite small, resulting in small changes to the filter weights.

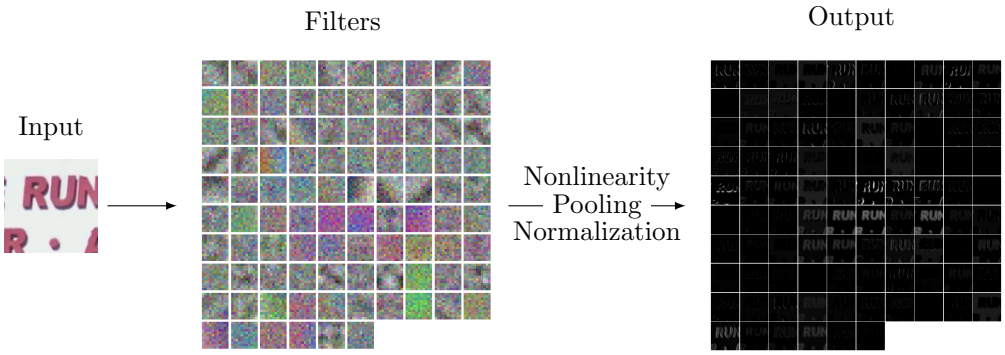
The filters for Network K, which was randomly initialized, appear as little more than random noise. The filters do, however, perform their task well, as they contributed to a 0.843 accuracy on the test dataset. One reason for the lack of structures in the filters of this network could be the fact that not enough data was available for clear structures to emerge during training.

It should also be noted that the filters in all the subsequent layers play a big role in the classification of an image window. The visualizations in Figure 5.2 simply serve to display how the networks finetuned from a previous network seem to retain the structures present in the first layer of that network, while the filters of the randomly initialized networks do not exhibit similar structures.

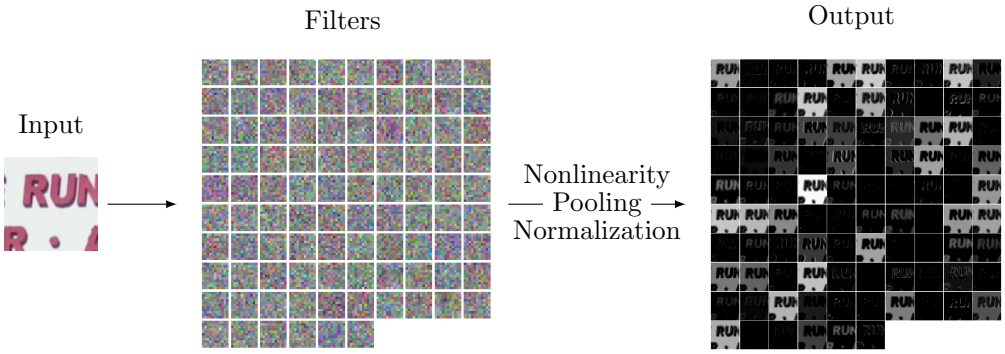
5.1.8 Incorrect classifications

The results in Table 5.1 indicate that Network N achieves the best accuracy on our test set. This network does, however, still incorrectly classify many text and not-text samples in the set. Out of the 5,207 not-text samples in the test set, 1,010 are classified as text, while 501 out of the 5,488 text samples are classified as not-text. It is not surprising that the number of false positives is higher than the number of false negatives, since there exists greater variance in the not-text class. Moreover, the background may contain structures with text-like features that are incorrectly classified. A few typical examples of incorrect classifications are shown in Figure 5.3.

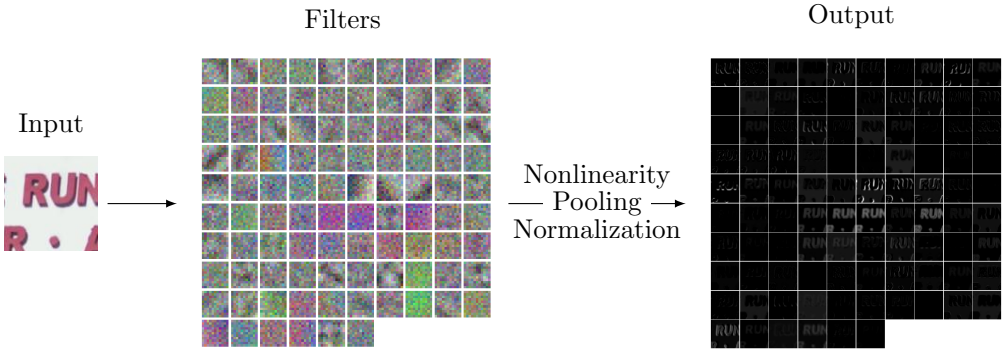
Due to the nature of neural networks and the fact that the feature extraction is not engineered according to what we perceive as good descriptions of text, it can be quite difficult to determine why these incorrect classifications occur. Because of this level of obscurity, much of our reasoning in this section



(a) Network H (random initialization, trained on synthetic dataset).



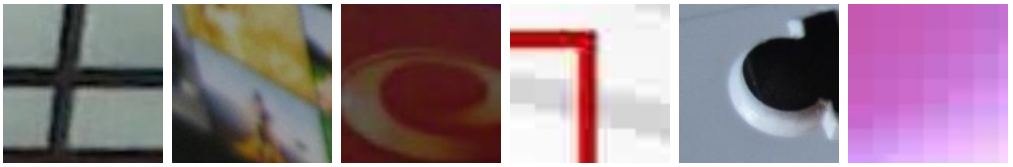
(b) Network K (random initialization, trained on largest natural dataset).



(c) Network N (initialized to Network H, trained on largest natural dataset).

Figure 5.2: Visualization of the filters in the first convolutional layer of networks with the ImageNet structure. The corresponding output for when each filter is applied to the specific input image window is also given.

(a) False negative classifications.



(b) False positive classifications.

Figure 5.3: Examples of windows from the test set that were incorrectly classified by Network N.

is mere speculation. There does, however, seem to be some shared properties between windows that are incorrectly classified.

Many of the false negative classifications can be attributed to poorly positioned windows. It is especially hard to recognize examples of text if a window covers only part of a large character. Furthermore, windows that contain extremely small and almost indiscernible text are also often misclassified. Strange fonts that were not present in the training set also tend to be incorrectly classified, while background noise could also interfere.

Some of the false positive classifications exhibit text-like structures, such as lines with sharp angles or structures that have uniform strokes. Window frames, for example, tend to be classified as text.

There are a few examples that defy explanation. Some windows that contain legible characters in a generic font are classified as not-text, while some windows without any clear structures are classified as text. Although it is difficult to determine the cause of these misclassifications, it is likely due to the nature of the training process, where, to generalize the network for a greater variety of text and background, some changes to features might make the network susceptible to incorrect classifications for specific cases.

5.2 Text detection in full images

In this section we present results from identifying regions of text in full images. Tests in this section were performed on the ICDAR dataset (Lucas *et al.*, 2003), to see how our trained networks generalize to data from a source never seen before. We focus on precision, recall and the f-measure as measurements for the accuracy, as opposed to the combined true-positive and true-negative rate. This is due to the large number of not-text pixels present in full images.

We measure precision, recall and f-measure of the saliency maps generated

by our system on a per pixel basis. The file describing the annotations of a given test image is used to create a saliency map in which all pixel falling within a bounding box are set to text, while the others are set to not-text, effectively generating a "perfect" saliency map. This map is then compared to the one generated by our system to determine true positives and negatives as well as false positives and negatives, which are in turn used when calculating precision, recall and the f-measure.

Similar to the discussion in Section 2.3, the precision, recall and f-measure are determined as follows:

$$\text{Precision} = \frac{T_p}{T_p + F_p}, \tag{5.1a}$$

$$\text{Recall} = \frac{T_p}{T_p + F_n}, \tag{5.1b}$$

$$\text{f-measure} = \left(\frac{0.5}{\text{Precision}} + \frac{0.5}{\text{Recall}} \right)^{-1}, \tag{5.1c}$$

where T_p, F_p, F_n refer to the number of true positive, false positive and false negative pixels respectively.

To give some indication of how the different networks performed during full image classification, Figure 5.4 presents saliency maps for two different images generated by Networks H, K and N. These saliency maps were generated on a single scale, but clearly show the differences between the networks. Network H, which was trained on synthetic data, seems to struggle with detecting all the text, and finds only larger words or characters. Randomly initialized Network K performs much better but lacks a bit in precision by presenting many false positives. Finally, the finetuned Network N seems to offer the best performance, with its saliency map more centred around text.

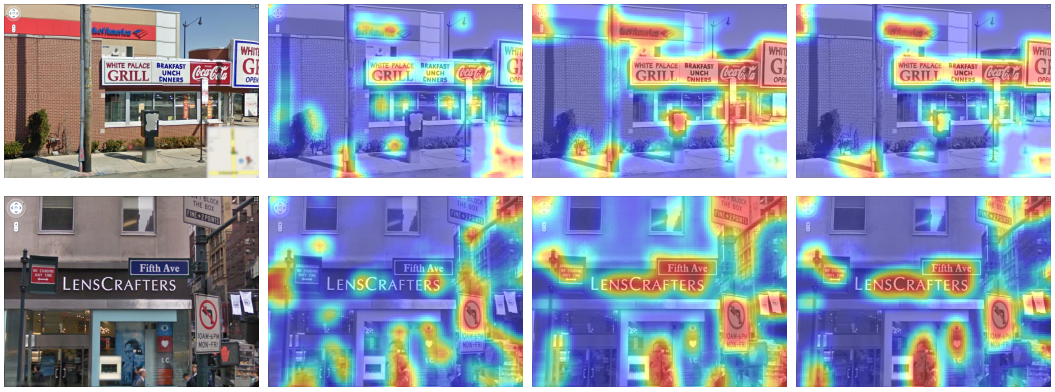


Figure 5.4: Comparison between saliency maps generated by different networks, using a sliding window classifier over a single scale. These maps are visualized in the standard jet colour scheme, and blended over the original image for visual orientation. The first map was generated by Network H, the second by Network K and the third by Network N.

5.2.1 Execution times

One of the drawbacks to using a sliding window, texture based approach is that it may be more time consuming than a connected component based approach (like the stroke width transform). Since the larger image is broken down into multiple windows that each needs to be classified, the execution time is directly related to the number of windows used. As a result, generating saliency maps from smaller windows also tend to take longer.

In the rest of this section we focus only on Network N, as it beat all the others in terms of window classification accuracy. On average, it takes this network around 2.8 milliseconds to classify a single window. In addition to the classification time, there is some extra overhead for resizing the given image, to detect text at different scales. Timing results at different scales, as well as the average number of windows required at each scale for both our natural training set and the ICDAR training set are presented in Table 5.3.

Images from the ICDAR training set lead to more windows than those from our training set, due to differences in aspect ratios (window sizes are based on the width of an image).

Scale	Scaling factor	Natural training set		ICDAR training set	
		#windows	Time (s)	#windows	Time (s)
1	1	575	1.627	959	2.650
2	1.25^{-1}	324	0.930	536	1.489
3	1.25^{-2}	179	0.529	290	0.815
4	1.25^{-3}	95	0.300	154	0.441
5	1.25^{-4}	52	0.180	81	0.239
6	1.25^{-5}	25	0.103	40	0.124
7	1.25^{-6}	13	0.069	19	0.066
8	1.25^{-7}	6	0.046	9	0.037

Table 5.3: The average number of windows and time required to process a full image from either our natural training dataset (from which the windows in `natural_1` were sampled) or the ICDAR training set, at different scales. The scaling factor indicates the factor by which the original image was resized.

5.2.2 Image pyramids

To compute the final segmentation of an image into text and not-text regions, the layers of an image pyramid (i.e. the saliency maps obtained from different scales of the image) are combined through majority voting. In this process, a pixel is only set to be text if the corresponding pixels in more than half of the saliency maps generated at different scales are also classified as text.

We considered various combinations of scales, and their performance on a training set, in an effort to find one that can be tested on the holdout set.

The precision, recall and f-measures were taken for each combination on both the ICDAR training set (that the network has not seen) as well as images in our own natural training set (that the network was trained on). Results are presented in Tables 5.4 and 5.5. Taking these results into consideration, we opted to use the first three scales in further testing. We stress that this decision was reached through consideration on training data, not test data.

The apparently low performance metrics are due to the fact that we measure them on a per pixel basis, while the ground truth consists of bounding boxes. The performance of our system would then suffer particularly in the vicinity of larger text, where the sliding windows might detect the larger character, but achieve a lower accuracy since the bounding boxes contain a large amount of background pixels between the character pixels. An example of this problem is presented in Figure 5.5.

We also note that our system achieved higher accuracies on the ICDAR set than on its own training set. A possible reason for this is that the text in the ICDAR set is more clear, with less distortions and blurring, and the background less diverse.

There are of course many other ways in which the various scales can be combined, but as a proof of concept we found our majority voting strategy to perform quite well. The fact that our system performs well on the ICDAR dataset reinforces the idea that it would generalize to various previously unseen scenarios.

Scales	Precision	Recall	f-measure
1	0.304	0.669	0.372
1,2	0.340	0.561	0.378
1,2,3	0.296	0.615	0.362
1,2,3,4	0.320	0.511	0.343

Table 5.4: The recall, precision and f-measure achieved by different scale combinations on our natural training data. The scale indices refer to those used in Table 5.3. The results are obtained from a sliding window classification (using Network N) over 373 full images. Results for each image were combined using majority voting over the different scales.

Scales	Precision	Recall	f-measure
1	0.498	0.671	0.491
1,2	0.539	0.631	0.501
1,2,3	0.495	0.725	0.516
1,2,3,4	0.513	0.692	0.513

Table 5.5: The recall, precision and f-measure achieved by different scale combinations on the ICDAR training data. The results are obtained from 250 full images.

(a) Input image. (b) Ground truth. (c) Our saliency map.

Figure 5.5: An example to indicate why the performance metrics for our results might seem low. Although our result seems to match the text in the image quite well, the use of the ground truth bounding box as a measurement of accuracy does not reflect that.

5.3 Comparison to other techniques

It is quite difficult to compare our results to those obtained by the techniques discussed in Chapter 2. The difficulty arises due to the fact that our accuracies are computed on a per pixel basis, while those other techniques base their accuracies on comparisons of their predicted bounding boxes with the ground truth bounding boxes. There are a number of reasons why this difference could be problematic.

Since the sizes of images in the test set can differ a lot, we decided to compute the precision and recall for each image and present the average of these measures as our final results. This means that a single incorrectly classified area in one image could have a much larger negative effect on the average precision and accuracy than a single incorrect bounding box would have. Furthermore, some of the other techniques accepted a bounding box as correct if it had a 0.5 overlap with the ground truth bounding box. Where some bounding boxes might receive a precision and recall value of 1.0, a similar region on a per pixel basis might receive precision and recall values as low as 0.5.

Due to time constraints we did not implement the other text detection techniques, but simply compare in Table 5.6 the accuracies obtained on the ICDAR dataset using our best preforming method (Network N and a combination of scales 1, 2 and 3) with those reported by the authors of the other techniques. Besides the fact that the bounding box and per pixel differences might affect the accuracies, it is also important to note that we did not use the ICDAR training set during the training of our neural network. As previously mentioned, this dataset was purposefully omitted to allow us to determine how well our system would generalize to previously unseen scenarios.

The recall achieved by our system on full image segmentation is comparable to the other systems, although the precision and f-measure are quite low. This could be improved by simply applying a dilation and erosion algorithm to smooth edges and remove areas that would normally be considered too small to

Method	Precision	Recall	f-measure
Yangxing *et al.* (2006)	0.64	0.67	0.65
Epshtein *et al.* (2010)	0.73	0.60	0.66
Chen *et al.* (2011)	0.73	0.60	0.66
Yao *et al.* (2012)	0.69	0.66	0.67
Yin *et al.* (2014)	0.67	0.85	0.75
Jaderberg *et al.* (2016)	0.96	0.85	0.90
Our method (full images)	**0.42**	**0.70**	**0.53**
Our method (sampled windows)	**0.87**	**0.69**	**0.77**

Table 5.6: Comparison between precision, recall and f-measure on the ICDAR test set. Since these metrics favour bounding boxes, which our method does not compute, we also include results from classifying 178,243 windows (89,153 text and 89,090 not-text) randomly sampled from both the ICDAR training and test sets.

qualify as text. Although we experimented with these functions, we ultimately decided not to include it in our results, since it might obscure the difference between the accuracy of the network and the accuracy due to post-processing.

As previously stated, these results are affected quite severely by the fact that the ground truth annotations come in the form of bounding boxes, while our system outputs a per pixel classification. We experimented with a number of methods to extract bounding boxes from a saliency map. This turned out to be quite a complicated problem, however, as detected regions of a single word might not be strongly connected while separate words might seem connected, the boundaries of detected areas might be irregular, or words might only be partially detected. Due to these difficulties, and the heuristic parameter tuning that seemed necessary, we opted to exclude bounding box extraction. Moreover, it can become uncertain whether the bounding box extraction method or the neural network itself is responsible for a measured text detection accuracy, which conflicts with the objectives of this study.

In an effort to get a better sense of how our classifier generalizes to previously unseen data, we randomly sampled 178,243 windows from the ICDAR dataset and classified each of them with our trained network. The result is shown in the last row of Table 5.6.

5.4 Qualitative results

Finally, for qualitative purposes, we present saliency maps generated by our system for various test images. The saliency maps shown in this section are simply averages over the first three scales, without applying a threshold or majority voting. The results, all generated with the use of Network N, include some of the best as well as worst detections.

Figure 5.6 contains images from the MSRA-TD500 dataset, Figure 5.7 contains images that were taken from the cnn.com livestream, Figure 5.8 contains images from the ICDAR test dataset and Figure 5.9 contains images from the Google Street View Text dataset.

Figure 5.6: Saliency maps generated for test images from the MSRA-TD500 dataset, showing examples of upside down text (left), non-horizontal text lines (middle) and a correctly classified uncommon font along with a large incorrectly detected region (right).

Figure 5.7: Saliency maps generated for test images from the cnn.com livestream, showing incorrectly classified structures along with accurate text detections (left), text of different colours and sizes (middle) and an incorrect background classification (right).

Figure 5.8: Saliency maps generated for images from the ICDAR test set, showing examples of a small area of text in an image (top left), non-aligned text (top middle), shadows over text (top right), incorrectly classified noise (bottom left), different fonts and colours as well as partially obscured text (bottom middle) and incorrectly classified structures (bottom right).

Figure 5.9: Saliency maps generated for test images from the Google Street View Text dataset, showing examples of text of different fonts and colours as well as partially obscured text (left), text of different sizes and slightly curved (middle) and missed words as well as incorrect classification of a background structure (right).

Chapter 6

Conclusions and future work

6.1 Conclusions

The goal of this study was to determine the viability of convolutional neural networks for text detection in natural images. We chose these networks because of the great potential they have shown in tasks involving object recognition from natural images as well as text recognition. We trained multiple networks for the task of classifying smaller areas from a given image as text or not-text. A sliding window approach could then be taken to scan the full image. The various trained networks differ in their structure as well as their initialization and the datasets they were trained on. We decided to employ the sliding window approach, since convolutional neural networks are well suited to the task of classification, rather than locating objects, and this approach would allow us to traverse a full image while searching for text.

Due to the nature of convolutional neural networks, large amounts of training data is required. We found it difficult to gather a sufficiently large annotated natural dataset, and so resorted to creating our own synthetic dataset. We blended synthetic text into natural images in order to account for more variance in the text, but found that training a neural network on this data did not allow it to generalize sufficiently well to the task of classifying natural data. We did find, however, that training a network on a large synthetic dataset and finetuning it on a much smaller natural dataset increased the accuracy of the network compared to networks trained solely on natural data.

By first training on a sufficiently large synthetic dataset, a network can learn certain structures that would otherwise not be found with a smaller training set. This is evidenced in Section 5.1.7 where the filters of the first convolutional layers from the synthetically trained and finetuned networks show clear structures, while the filters of the randomly initialized network look like little more than random noise.

Finally, we employed image pyramids in an attempt to detect text at different scales as well as to remove incorrect classifications. This was simply done

by extracting windows of different scales, classifying them to create multiple saliency maps and combining the saliency maps using majority voting.

Our results clearly indicate that a more complex network structure is required for better generalization. Networks with a smaller structure were also more likely to overfit on the training data, as evidenced by the high accuracies obtained on the training set compared to the accuracies obtained on the test set. Furthermore, larger datasets are also required for a network to better generalize. It is interesting to note, however, that the accuracies obtained when training on our medium sized training set compared to our large training set are similar, with a network trained on the medium dataset outperforming one trained on the large set in one instance. This is likely due to the fact that, apart from additional single character images contained in the large dataset, the two datasets share many of the source images from which windows were sampled.

Employing image pyramids did improve the overall accuracy, but only by a small amount. The reason for this small increase is likely due to the fact that windows used during training were taken from natural scenes with varying text sizes. In some cases, the image pyramids caused incorrect classifications in areas that were correctly detected at a single scale.

Compared to existing text detection methods, we achieved similar recall but lower precision. Both of these measures were, however, influenced by the manner in which results are compared to the ground truth. We decided to take a per pixel approach, which meant that correctly detected areas might have lower accuracies if the entire area included in the ground truth bounding box was not detected. Taking this into account, we believe that our method is viable for text detection, although the method for extracting text from the image should be investigated further.

The approach we took allowed us to move away from engineering specific features, in favour of allowing the network to look at raw data to determine which features best describe text. This move breaks away from the recent trend of the connected composed based approach, which relies heavily on thresholds to determine what is and is not text. We believe that our approach enables better generalization, especially in cases where the appearance of text might differ significantly from the training set. The fact that our network performed better on the ICDAR dataset, despite its training set not being used, supports this claim quite strongly.

There are, however, some disadvantages to using the sliding window approach. The first and most obvious is the time required. This is something that would not be improved much with improvements in technology, unless the classification could be parallelized. Since windows are used to traverse the entire image, a significant number of these windows need to be passed through a network. Even though classification happens in a fraction of a second, the sheer number of windows that need to be classified makes the process quite slow. Another negative aspect, compared to the connected components based

approach, is the fact that the text is not directly extracted from the image. We simply identify regions likely to contain text, while connected components based methods can extract characters and words for immediate recognition.

Overall, we believe that using convolutional neural networks for the task of text detection in natural images shows promise. Furthermore, using synthetic data as a replacement for natural data might not be sensible, but pre-training on the synthetic data and finetuning on natural data afterwards does seem to increase the accuracy achieved by a network. However, text is ideally suited for this task since it can easily be synthesized, and this approach might not work for other types of data. Finally, it is important to note that sufficiently large natural datasets might reduce the need for synthetic data, and enable the extraction of general features to successfully identify the underlying structures of text in natural images.

6.2 Future work

There are a number of things we can do to improve upon and add to our system. Some of the aspects that can be addressed include decreasing the computational time by identifying possible regions of text prior to classification by the network, extracting bounding boxes to increase the accuracy and allow for text recognition software, and adding our own text recognition capabilities to the system.

Although extracting possible regions of text might seem like an extra unnecessary step, it could greatly reduce the number of windows needed for classification. Depending on the implementation of this identification, it could also remove the need to classify windows at different scales, since the windows that cover the regions could be rescaled according to their size. By imposing certain restrictions like aspect ratio on the regions, we could further eliminate background structures that might exhibit text like appearance. This process could be done using simple bounding box generation through edge density boxes, by extracting easily computed connected components or a number of other methods.

Extracting bounding boxes can tie in with the above mentioned improvement, in the sense that bounding boxes can be extracted before classification starts. In this case an algorithm to centre and better fit the bounding box can be implemented after classification to allow for better word extraction. Additionally, bounding boxes can be extracted from the final saliency maps based on regions with a high probability of text. As mentioned in the previous point, this could reduce false positives by imposing certain restrictions on the dimensions and aspect ratios of the bounding boxes. Accurate bounding box extraction is also important to improve the accuracy of word recognition, which is the main reason for detecting text.

Finally, word recognition can be implemented in a number of different ways. Neural networks have shown great promise in this area, and would likely be our main focus for future work. Recognition would not only allow for the extraction of information from images, but could also improve the accuracy of the system by further eliminating false positive bounding boxes. Many text recognition systems can eliminate false positives, and neural networks are no exception, since a "not-text" category can simply be added to the classification.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M. *et al.* (2015). TensorFlow: large-scale machine learning on heterogeneous systems.
Available at: `http://tensorflow.org/`

Bergstra, J., Bastien, F., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., Desjardins, G., Warde-Farley, D., Goodfellow, I., Bergeron, A. and Bengio, Y. (2011). Theano: deep learning on GPUs with Python. In: *Conference on Neural Information Processing Systems Big Learn Workshop*.

Bissacco, A., Cummins, M., Netzer, Y. and Neven, H. (2013). PhotoOCR: reading text in uncontrolled conditions. In: *IEEE International Conference on Computer Vision*, pp. 785–792.

Bottou, L. (2012). Stochastic gradient descent tricks. In: *Neural Networks: Tricks of the Trade*, pp. 421–436. Springer.

Bradski, G. (1998). Real time face and object tracking as a component of a perceptual user interface. In: *IEEE Workshop on Applications of Computer Vision*, pp. 214–219.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698.

Chen, H., Tsai, S., Schroth, G., Chen, D., Grzeszczuk, R. and Girod, B. (2011). Robust text detection in natural images with edge-enhanced maximally stable extremal regions. In: *IEEE International Conference on Image Processing*, pp. 2609–2612.

Chen, X. and Yuille, A. (2004). Detecting and reading text in natural scenes. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 366–373.

Chen, Y. and Dunsmoir, J. (2009 February 24). System and method for automatic natural language translation of embedded text regions in images during information transfer. US Patent 7,496,230.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. In: *International Conference on Machine Learning*, pp. 160–167.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, vol. 12, no. 8, pp. 2493–2537.

De Campos, T., Babu, B. and Varma, M. (2009). Character recognition in natural images. In: *International Conference on Computer Vision Theory and Applications*, pp. 273–280.

Dollár, P., Appel, R., Belongie, S. and Perona, P. (2014). Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1532–1545.

Duchi, J., Hazan, E. and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, vol. 12, no. 7, pp. 2121–2159.

Epshtein, B., Ofek, E. and Wexler, Y. (2010). Detecting text in natural scenes with stroke width transform. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2963–2970.

Garcia, C. and Delakis, M. (2004). Convolutional face finder: a neural architecture for fast and robust face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1408–1423.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In: *International Conference on Artificial Intelligence and Statistics*, pp. 249–256.

Graves, A., Mohamed, A. and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In: *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649.

Grond, M., Brink, W. and Herbst, B. (2016). Text detection in natural images with convolutional neural networks and synthetic training data. In: *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference*, pp. 125–130.

Haritaoglu, I. (2001). Scene text extraction and translation for handheld devices. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 408–413.

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Jaderberg, M., Simonyan, K., Vedaldi, A. and Zisserman, A. (2016). Reading text in the wild with convolutional neural networks. *International Journal of Computer Vision*, vol. 116, no. 1, pp. 1–20.

Jaderberg, M., Vedaldi, A. and Zisserman, A. (2014). Deep features for text spotting. In: *European Conference on Computer Vision*, pp. 512–528.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T. (2014). Caffe: convolutional architecture for fast feature embedding.
Available at: `http://caffe.berkleyvision.org`

Krizhevsky, A., Sutskever, I. and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, pp. 1097–1105.

Lawrence, S., Giles, C., Tsoi, A. and Back, A. (1997). Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature*, vol. 51, no. 7553, pp. 436–444.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W. and Jackel, L. (1990). Handwritten digit recognition with a back-propagation network. In: *Advances in Neural Information Processing Systems*, pp. 396–404.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324.

LeCun, Y., Kavukcuoglu, K. and Farabet, C. (2010). Convolutional networks and applications in vision. In: *IEEE International Symposium on Circuits and Systems*, pp. 253–256.

Leshno, M., Lin, V., Pinkus, A. and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, vol. 6, no. 6, pp. 861–867.

Li, H., Doermann, D. and Kia, O. (2000). Automatic text detection and tracking in digital video. *IEEE Transactions on Image Processing*, vol. 9, no. 1, pp. 147–156.

Lienhart, R. and Wernicke, A. (2002). Localizing and segmenting text in images and videos. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 4, pp. 256–268.

Lucas, S., Panaretos, A., Sosa, L., Tang, A., Wong, S. and Young, R. (2003). ICDAR 2003 robust reading competitions. In: *International Conference on Document Analysis and Recognition*, pp. 682–687.

Maas, A., Hannun, A. and Ng, A. (2013). Rectifier nonlinearities improve neural network acoustic models. In: *International Conference on Machine Learning Workshop on Deep Learning for Audio, Speech and Language Processing.*

Matas, J., Chum, O., Urban, M. and Pajdla, T. (2004). Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, vol. 22, no. 10, pp. 761–767.

Neumann, L. and Matas, J. (2010). A method for text localization and recognition in real-world images. In: *Asian Conference on Computer Vision*, pp. 770–783.

Neumann, L. and Matas, J. (2012). Real-time scene text localization and recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3538–3545.

Rumelhart, D., Hinton, G. and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, vol. 323, pp. 533–536.

Ryzin, J.V. (1998 December 1). Automobile navigation system. US Patent 5,844,505.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. *et al.* (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, vol. 529, no. 7587, pp. 484–489.

Socher, R., Lin, C., Manning, C. and Ng, A. (2011). Parsing natural scenes and natural language with recursive neural networks. In: *International Conference on Machine Learning*, pp. 129–136.

Sutherland, I. and Hodgman, G. (1974). Reentrant polygon clipping. *Communications of the ACM*, vol. 17, no. 1, pp. 32–42.

Wang, K. and Belongie, S. (2010). Word spotting in the wild. In: *European Conference on Computer Vision*, pp. 591–604.

Wang, T., Wu, D., Coates, A. and Ng, A. (2012). End-to-end text recognition with convolutional neural networks. In: *International Conference on Pattern Recognition*, pp. 3304–3308.

Widrow, B. and Lehr, M. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442.

Wilson, D. and Martinez, T. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, vol. 16, no. 10, pp. 1429–1451.

Yangxing, L., Goto, S. and Ikenaga, T. (2006). A contour-based robust algorithm for text detection in color images. *IEICE Transactions on Information and Systems*, vol. 89, no. 3, pp. 1221–1230.

Yao, C., Bai, X., Liu, W., Ma, Y. and Tu, Z. (2012). Detecting texts of arbitrary orientations in natural images. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1083–1090.

Ye, Q., Huang, Q., Gao, W. and Zhao, D. (2005). Fast and robust text detection in images and video frames. *Image and Vision Computing*, vol. 23, no. 6, pp. 565–576.

Yi, C. and Tian, Y. (2011). Assistive text reading from complex background for blind persons. In: *International Workshop on Camera-Based Document Analysis and Recognition*, pp. 15–28.

Yin, X., Yin, X., Huang, K. and Hao, H. (2014). Robust text detection in natural scene images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 5, pp. 970–983.

Zeiler, M.D. (2012). ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zeiler, M.D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*.

Zitnick, C. and Dollár, P. (2014). Edge boxes: locating object proposals from edges. In: *European Conference on Computer Vision*, pp. 391–405.